Design of an Optimized Low Power Vedic Multiplier Unit for Digital Signal Processing Applications

A Project Report

submitted by

Nandita Bhaskhar
Roll no - EDM10B009

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Electronics Engineering (Design & Manufacturing)

Indian Institute of Information Technology Design & Manufacturing, Kancheepuram
Chennai

May, 2014

BONAFIDE CERTIFICATE

Certified that this project report titled **Design of an Optimized Low Power Vedic Multiplier Unit for Digital Signal Processing Applications** is the bonafide work of **Ms. Nandita Bhaskhar** who carried out the research under my guidance. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

(Dr. Binsu J Kailath)
Project GUIDE
Assistant Professor
IIITD&M, Kancheepuram
Chennai 600-127

Place: Chennai
Date: 26th May, 2014

ACKNOWLEDGEMENTS

I am deeply grateful to my guide and mentor, Prof. Dr. Binsu J Kailath, for her invaluable guidance, support and encouragement, without which this project could not have been completed successfully. Her incredible patience and helpful advice, combined with her care and friendship made my project work truly satisfying and pleasurable. I am indeed very privileged to have got this opportunity to work with her.

I would also like to thank all the faculty members in the Electronics Department for their timely suggestions, positive criticism and insightful remarks which encouraged me to give more depth to my project. I am very grateful to my lab technicians and other staff members for obligingly acquiescing to all my requests. Their assistance and support truly made my work easier. I'd like to extend my gratitude to the Computer Science Department for letting me use the labs full time. And I thank the Director for giving me this great opportunity.

The acknowledgements wouldn't be complete without mentioning my family and friends. I'd like to thank my parents and sister for giving me their love & support despite all my grouchiness and I'd like to extend it to my close friends who put up with my peeves very cheerfully. And I am forever indebted to a very special friend of mine for encouraging me throughout & for coming to my rescue especially when I needed it. I couldn't have done my report without you.

Nandita Bhaskhar

ABSTRACT

Digital multipliers play a crucial role in various Digital Signal Processing units. They carry the major responsibility of power expenditure in the system and ultimately determine its speed. As a result, it is always beneficial to develop high performance, low power multipliers.

Vedic Mathematics is a set of mathematical rules, derived from ancient Indian scripts that makes arithmetic calculations extremely fast and simple. There are 16 rules or *Sutras* expounded in Vedic Mathematics. This report presents novel designs of a multiplier based on the Vedic Sutras on multiplication - *Urdhva Tiryakbhyam* and *Nikhilam.*

The objective of this report is to develop an optimum Vedic multiplier for $128-bit$ inputs. Various fresh algorithms and strategies based on both the sutras as well as a combination of them are propounded and implemented to develop the most optimum multiplier in terms of power consumption, delay and area occupied. The proposed multipliers are designed for synthesis using Carry Look Ahead Adders and compared with other existing multipliers like Array, Booth and Modified Booth multipliers and the performances are evaluated. The design of a novel integrated $128-bit$ multiplier is presented along with its mathematical analysis of power and is validated by its optimum time delay, area occupancy and minimum power consumption of 4.971 ns, 75013 nm^2 and 28.73 mW respectively.

KEYWORDS: Integrated digital multiplier, Vedic Mathematics, Vedic Sutras, Urdhva Tiryakbyam, Nikhilam, low power, optimum area, reduced time delay, $128-bit$, Array, Booth, Modified Booth

Contents

Acknowledgements **i**

Abstract **ii**

List of Tables **vi**

List of Figures **viii**

List of Abbreviations **ix**

List of Notations **x**

1 Introduction **1**
1.1 Vedic Mathematics - An Overview 3
1.2 Motivation 3
1.3 Objective 5
1.4 Organization of the Report 6

2 Literature Review **7**

3 Theoretical Background – Algorithms **10**
3.1 Urdhva Tiryakbhyam 10
3.2 Nikhilam 12
3.3 Karatsuba-Ofman Algorithm 13
3.4 Carry Look Ahead Adder 15

4 Design & Implementation **19**
4.1 Synthesizable Code for Hardware Efficiency 19
4.2 Vedic UT 20
4.3 Scaling – The overall plan 20
4.3.1 Where have the optimizations taken place? 22
4.4 UT Squaring 22
4.4.1 Scaling – Adapted for Squares 22
4.5 Proposed Design D 23
4.6 Thresholding Nikhilam 24
4.7 Successive Nikhilam 25

5 Sampoornam – the Proposed Integrated Multiplier 28
5.1 Sampoornam: Specifications . . . 28
5.2 Sampoornam: Logic . . . 29
5.3 Sampoornam: Implementation . . . 30
5.4 Sampoornam: Advantages . . . 31

6 Simulations and Results 32
6.1 Vedic UT . . . 32
6.1.1 RTL Schematics . . . 32
6.1.2 RTL Results . . . 36
6.2 Vedic Square . . . 36
6.2.1 RTL Schematics . . . 36
6.2.2 RTL Results . . . 38
6.3 Design D . . . 38
6.3.1 RTL Results . . . 38
6.4 Nikhilam GG . . . 38
6.4.1 RTL Schematics . . . 38
6.4.2 RTL Results . . . 40
6.5 Nikhilam - SG . . . 40
6.5.1 RTL Schematics . . . 40
6.5.2 RTL Results . . . 42
6.6 Nikhilam SS . . . 42
6.6.1 RTL Schematics . . . 42
6.6.2 RTL Results . . . 43
6.7 Logic Block . . . 44
6.7.1 RTL Schematics . . . 44
6.7.2 RTL Results . . . 44
6.8 CLA Blocks . . . 45
6.8.1 RTL Analysis . . . 45
6.9 Test Bench Output . . . 45

7 Inferences 47
7.1 Comparison –
Vedic UT, Vedic Square & Design D . . . 47
7.2 A look at the 3 cases of Nikhilam . . . 49
7.3 Analytical Comparisons with the Vedic UT . . . 50
7.4 Analysis of the Logic Block and CLA Adder . . . 51
7.4.1 Logic Block . . . 51
7.4.2 CLA Adder . . . 51

8 Comparison with Existing Work 53

9 Power Analysis 55
9.1 Power analysis – Sampoornam . . . 56

10 Conclusion **58**
10.1 Future Scope 59

Bibliography **60**

Appendix A - Vedic Sutras **62**

Appendix B - Cadence Encounter **63**

List of Tables

3.1 Adder Comparison – N: input size, k: group size 18

4.1 Thresholds for various multipliers 24
4.2 Multiplication of $m \times n$, base r 24
4.3 Multiplication of $m \times n$, base r 25
4.4 Multiplication of $m \times n$, base r 25
4.5 Nikhilam – 1111×1111, base = 1000 26
4.6 Total No. of Adders Required – Successive Nikhilam 27

5.1 Designs and their input constraints 29
5.2 Priority Encoder – Output 30

6.1 Vedic UT 36
6.2 Vedic Square 38
6.3 Design D 38
6.4 Nikh GG 40
6.5 Nikh SG 42
6.6 Nikh SS 43
6.7 Logic 44
6.8 CLA Analysis 45

7.1 Vedic UT to Vedic Square: % reduction 50
7.2 Vedic UT to Design D: % reduction 50
7.3 Vedic UT to Nikhilam (maximum): % reduction 50

8.1 As reported in Literature 53
8.2 Vedic UT 54

9.1 Submodules & their Power consumption 55
9.2 Submodules – Sampoornam 56

List of Figures

1.1 A typical Digital Processing System . . . 1
1.2 A MAC unit . . . 2
1.3 Array Multiplier – Algorithm . . . 4
1.4 Booth Multiplier – Algorithm . . . 5

3.1 UT for a 3×3 Decimal Multiplication . . . 10
3.2 UT for a $4 - bit$ multiplication . . . 11
3.3 Nikhilam Example in Decimal Numbers . . . 12
3.4 Standard Scaling Multiplication Algorithm . . . 14
3.5 Karatsuba-Ofman Algorithm . . . 14
3.6 Partial Full Adder (PFA) . . . 15
3.7 $4 - bit$ CLA . . . 16
3.8 $16 - bit$ CLA adder from $4 - bit$ CLA adders . . . 16
3.9 Critical Path of a $16 - bit$ CLA . . . 17

4.1 Structural Programming . . . 19
4.2 Multiplication of two $2n - bit$ numbers . . . 20
4.3 $2n - bit$ Squaring Unit . . . 22

5.1 Sampoornam Logic – Flowchart . . . 29
5.2 Sampoornam – Implementation . . . 30

6.1 $2 - bit$ Vedic UT . . . 32
6.2 $4 - bit$ Vedic UT . . . 32
6.3 $8 - bit$ Vedic UT . . . 33
6.4 $16 - bit$ Vedic UT . . . 33
6.5 $32 - bit$ Vedic UT . . . 34
6.6 $64 - bit$ Vedic UT . . . 34
6.7 $128 - bit$ Vedic UT . . . 35
6.8 $2 - bit$ Vedic Square . . . 36
6.9 $4 - bit$ Vedic Square . . . 36
6.10 $8 - bit$ Vedic Square . . . 37
6.11 $16 - bit$ Vedic Square . . . 37
6.12 $8 - bit$ Nikh GG . . . 38
6.13 $16 - bit$ Nikh GG . . . 39
6.14 $32 - bit$ Nikh GG . . . 39
6.15 $64 - bit$ Nikh GG . . . 39

6.16 $8 - bit$ Nikh SG 40
6.17 $16 - bit$ Nikh SG 41
6.18 $32 - bit$ Nikh SG 41
6.19 $8 - bit$ Nikh SS 42
6.20 $16 - bit$ Nikh SS 42
6.21 $32 - bit$ Nikh SS 43
6.22 $8 - bit$ Nikh SS 44
6.23 $64 - bit$ Vedic UT 45
6.24 $128 - bit$ Vedic UT 46

7.1 Power Consumption in nW 47
7.2 Area occupied in nm^2 48
7.3 Worst Path Delay in ps 48
7.4 Comparison between the 3 cases of Nikhilam 49
7.5 Logic Modules – Variation of Parameters 51
7.6 CLA Adder – Variation of Parameters 52

List of Abbreviations

DSP	Digital Signal Processing
MAC	Multiplication Accumulation Unit
UT	Urdhva Tiryakbhyam
Nikh	Nikhilam
CLA	Carry Look Ahead
RCA	Ripple Carry Adder
CSKA	Carry Skip Adder
CSLA	Linear Carry Select Adder
HDL	Hardware Description Language
av	Average
dev	Deviation
GG	Greater than Base, Greater than Base
SG	Smaller than Base, Greater than Base
SS	Smaller than Base
RTL	Register Transfer Logic
nW	nano Watts
ps	pico seconds
nm^2	nano square meters
DNG	Data not given
MSB	Most significant Bit
LSB	Least Significant Bit
ASIC	Application Specific Integrated Circuit

List of Notations

$\otimes$	XOR
#	number of
r'b0	$r - bit$ number is 0
$A[(n-1):0]$	$n - bit$ number with i^{th} bit given by $A[i]$
$\rightarrow$	implies / leads to
$\overline{a}$	a complement

Chapter 1

Introduction

Digital signal processing (DSP) is firmly being established as an extremely vibrant and vital field in the Electronics industry. The past few decades have seen an exponential growth in the number of products and applications that involve DSP, with a wide reach into diverse domains such as audio signal processing, digital image processing, video compression, speech processing, speech recognition, digital communications, RADAR, SONAR, financial signal processing, seismology and even biomedicine. Especially since computers have evolved into powerful machines capable of high computational complexity, almost all the signal processing takes place in the Digital Domain.

Frequently used algorithms include the Convolution operation, Finite Impulse Response (FIR) Filter, Infinite Impulse Response (IIR) Filter, and Fast Fourier Transform (FFT), all of which require intensive computation. DSP algorithms generally require a large number of mathematical operations to be performed quickly and repeatedly on a series of incoming data. The signals are constantly converted from analog to digital, digitally manipulated, and then converted back to analog.

Figure 1.1: A typical Digital Processing System

Most general–purpose microprocessors and operating systems can execute DSP algorithms successfully, but consume more power and occupy a larger area which is not suitable for most portable applications like those on mobile phones, biomedical devices, etc. A specialized digital signal processor, the **Digital Signal Processor (DSP processor)**, having different architectures and features optimized specifically for digital signal processing, is hence preferred. This will tend to provide a lower-cost solution, with better performance, lower latency and lesser power consumption. Thus, the efficiency in the design of the underlying hardware in the DSP processors will reflect in the performance of the applications.

One of the most important hardware structures in a DSP processor is the **Multiply-Accumulate (MAC)** unit. A conventional MAC unit consists of an $n - bit$ multiplier, the output of which is added to/subtracted from the contents of an Accumulator that stores the result. Thus, the MAC unit implements functions of the type $A + BC$. The ability to compute with a fast MAC unit is essential to achieve high performance in many DSP algorithms, and which is why there is at least one dedicated MAC unit in all of the modern commercial DSP processors.

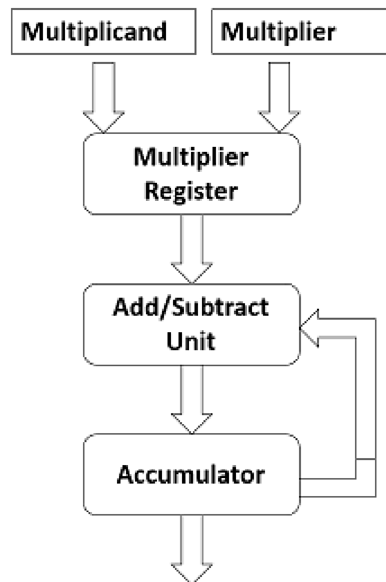

Figure 1.2: A MAC unit

Hence as it can be observed, **Digital Multipliers** are the core components of all MAC units and hence all DSP processors. The multiplier lies in the ***Critical Delay Path*** and ultimately determines the performance of any algorithm in the processor. Currently, multiplication time is still the major factor in determining the instruction cycle time of a DSP chip apart from contributing to the bulk of its power expenditure. Since multiplication drains power quickly and dominates the ***execution time*** of most DSP algorithms, there is a need for **Low–Power, High–Speed Multipliers**. In this concern, design of efficient multipliers has long been a topic of interest to digital design engineers.

The other function that a MAC unit inherently performs is the addition operation. It is one of the most essential operations in the instruction set of any processor. Other instructions such as subtraction and multiplication employ addition in their operations, and their underlying hardware is primarily dependent on the addition hardware. Hence the performance of a design will be often be limited by the performance of its adders. It is therefore as important to choose the correct adder to implement in a design as it is to choose a multiplier because of the many factors it affects in the overall chip.

The main expected features of any DSP block, be it an adder or a multiplier, are *speed*, *accuracy* and *easy integrability*. A number of interesting algorithms have been reported in literature, each offering different advantages and having trade-offs in terms of speed, circuit complexity, area and power consumption, forming an active area of research.

1.1 Vedic Mathematics - An Overview

Vedic Mathematics is the name given to a set of rules derived from Ancient Indian Scriptures, elucidating different mathematical results and procedures in simple and understandable forms. The word ***Vedic*** is derived from the word ***Veda*** which means the store–house of all knowledge.

It is claimed to be a part of the ***Sthapatya Veda***, a book on civil engineering and architecture, which is an ***Upaveda*** (supplement) of the ***Atharva Veda***. It covers explanations of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus.

The beauty of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome-looking calculations in conventional mathematics to very simple ones. This is because the Vedic formulae are claimed to be based on the natural principles on which the human mind works.

Vedic mathematics is mainly based on 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry, etc. Of these, there are two Vedic Sutras meant for quicker multiplication. They have been traditionally used for the multiplication of two numbers in the decimal number system. They are –

1. ***Nikhilam Navatashcaramam Dashatah***: All from 9 and last from 10
2. ***Urdhva Tiryakbhyam***: Vertically and crosswise

1.2 Motivation

Multiplication involves two basic operations – the generation of partial products and their accumulation. Clearly, a smaller number of partial products reduces the complexity, and, as a result, reduces the partial products accumulation time.

When two $n-bit$ numbers are multiplied, a $2n-bit$ product is produced. Previous research on multiplication, i.e. shift and add techniques, focused on multiplying two $n-bit$ numbers to produce n partial products and then adding the n partial products to generate a $2n-bit$ product. In which case, the process is sequential and requires $\boldsymbol{n}$ processor cycles for an $n \times n$ multiplication. Advances in VLSI have rendered ***Parallel Multipliers*** – fully combinational multipliers, which minimize the number of clock cycles/steps required, feasible.

Two most common multiplication algorithms followed in the digital hardware are the ***Array*** multiplication algorithm and ***Booth*** multiplication algorithm. The computation time taken by the array multiplier is comparatively less because the partial products are calculated independently in parallel. The delay associated with the array multiplier is the time taken by the signals to propagate through the gates that form the multiplication array. In the case of Booth multiplication algorithm, it multiplies two signed binary numbers in two's complement notation. Andrew Donald Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. It is possible to reduce the number of partial products by half, by using the technique of Radix-4 Booth recoding. But both do have their own limitations. The search for a new design of a multiplier which will radically improve the performance is always on.

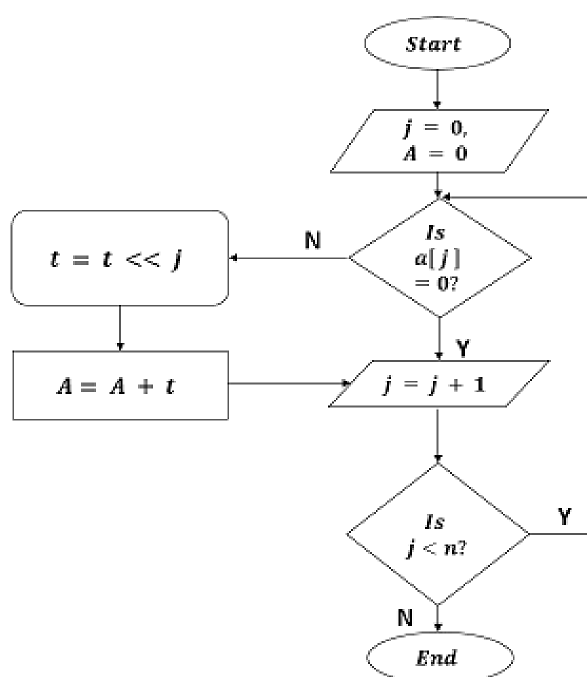

Figure 1.3: Array Multiplier – Algorithm

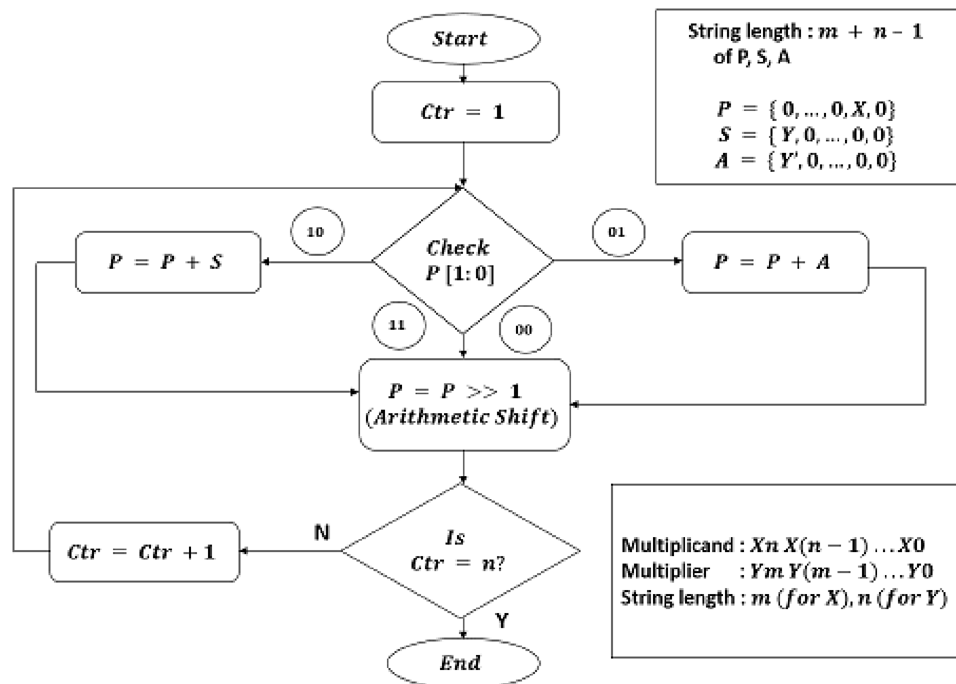

Figure 1.4: Booth Multiplier – Algorithm

The ***main motivation*** of this project is to make use of the simplicity of the Vedic sutras and adapt it for binary arithmetic to get an efficient and optimum digital multiplier fulfilling the demands of the growing technology. This, if implemented correctly, has the capability to reduce the computational time of DSP applications to a fraction of what it is today and revolutionize the standard power consumption of DSP chips. Fortunately or unfortunately, the potential of the Vedic Algorithms has remained untapped and unplundered for long. It would be a source of pride to prove that the Indian–originated methods can surpass the existing algorithms and to use them widely in various applications for the benefit of the industry.

1.3 Objective

The aim of this project is to design and implement an optimized digital multiplier which will multiply two real integers for DSP applications incorporating Vedic Multiplication principles.

The goals are:

- To reduce the **computational time**
- To optimize the **area** occupied
- To minimize the **power** consumed

- To realize it for **128 bits**
- To develop novel algorithms for obtaining a highly optimum multiplier
- To design and implement an integrated multiplier which will decide the algorithm to be used depending on the given inputs
- To finalize on an optimum adder and use it to implement all the addition operations

1.4 Organization of the Report

Chapter 1, the **Introduction** describes the need for efficient multipliers followed by a brief overview of Vedic Sutras. It illustrates the motivation behind taking up this project and states the objectives.

Chapter 2, **Literature Review**, deals with all the multiplication and addition schemes reported in literature. It puts forward the concepts already proposed on related areas in journals and conferences.

Chapter 3, **Theoretical Background - Algorithms** presents the Vedic Multiplication Algorithms – Urdhva Tiryakbhyam and Nikhilam, in detail, then proceeds with an explanation of the Karatsuba-Ofman Algorithm and concludes by justifying the selection of the Carry Look Ahead (CLA) Adder.

Chapter 4, **Design & Implementation**, expounds on the main work done in the project, beginning with an explanation of the modular hierarchy followed in this project, and the moving on to the actual logic employed in writing ***synthesizable*** code for each module and finally concluding with an emphasis on Structural modelling as compared to behavioural modelling.

Chapter 5, **Sampoornam – the Proposed Integrated Multiplier** presents a novel multiplier designed such that the logic decides which algorithm is to be used based on the input along with techniques employed to build this optimum, smart multiplier.

Chapter 6 gives all the **Simulations and Results** while Chapter 7 presents the **Inferences** of the results.

Chapter 8 is dedicated to **Power Analysis**, which gives a true picture of the power consumed in real time, based on mathematical analyses.

Finally, Chapter 9, the **Conclusion** completes the report by summarizing the work and considering the future scope .

Chapter 2

Literature Review

Here, a brief summary of the work that has already been done in this field with Vedic Multipliers is presented. A few results are noted down for comparison later.

The implementation of an $8-bit$ Vedic multiplier enhanced in terms of propagation delay when compared with conventional multipliers like Array multiplier, Braun multiplier, Modified Booth multiplier and Wallace tree multiplier has been given by *Pavan Kumar U.C.S, et al, 2013*. Here, they have utilized an $8-bit$ barrel shifter which requires only one clock cycle for n number of shifts. The design could achieve propagation delay of **6.781 ns** using barrel shifter in base selection module and multiplier.

S. Deepak, et al, 2012, have proposed a new multiplier design which reduces the number of partial products by 25 %. This multiplier has reported to have been used with different adders available in literature to implement multiplier accumulator (MAC) unit and parameters such as propagation delay, power consumed and area occupied have been compared in each case. From the results, Kogge Stone adder was been chosen as it was claimed to have provided optimum values of delay and power dissipation. The results obtained have been compared with that of other multipliers and it has been reported that the proposed multiplier has the lower propagation delay when compared with Array and Booth multipliers.

A high speed complex multiplier design (ASIC) using Vedic Mathematics has also been reported by *Prabir Soha, et al, 2011.* A complex number multiplier design based on the formulas of the ancient Indian Vedic Mathematics, was said to have been implemented in Spice spectre and compared with the mostly used architecture like distributed arithmetic, parallel adder based implementation, and algebraic transformation based implementation. It claims to have combined the advantages of the Vedic mathematics for multiplication which encounters the stages and partial product reduction. The proposed complex number multiplier has been reported to offer 20% and 19% improvement in terms of propagation delay and power consumption respectively, in comparison with parallel adder based implementation. The corresponding improvement in terms of delay and power was reported to be 33% and 46% respectively, with reference to the algebraic transformation based implementation.

Mohammed Hasmat Ali, et al, 2013, have presented a detailed study of different multipli-

ers based on Array Multiplier, Constant coefficient multiplication (KCM) and multiplication based on Vedic Mathematics. The reported multipliers have been coded in Verilog HDL (Hardware Description Language) and simulated in ModelSimXEIII6.4b and synthesized in EDA tool Xilinx ISE12. All multipliers are compared based on LUTs (Look up table) and path delays. Results report that Vedic Urdhva Tiryakbhyam sutra is the fastest Multiplier with least path delay. The computational path delay for proposed 8×8 bit Vedic Urdhava Tiryakbhyam multiplier was reported to be **17.995 ns**.

Karatsuba-Ofman algorithm has been reported to have been used by *M.Ramalatha, et al, 2009*, in the implementation of an efficient Vedic multiplier which is meant to have high speed,less complexity and consuming less area. Also after using this multiplier module a Vedic MAC unit was constructed and both these modules were integrated into an arithmetic unit along with the basic adder subtractor.

A generalized algorithm for multiplication has been reported by Ajinkya Kala, 2012, through recursive application of the Nikhilam Sutra from Vedic Mathematics, operating in radix - 2 number system environment suitable for digital platforms. Statistical analysis has been carried out based on the number of recursions profile as a function of the smaller multiplicand. The proposed algorithm was claimed to be efficient for smaller multiplicands as well, unlike most of the asymptotically fast algorithms. It was implemented for same sized inputs but an algorithm was presented which could be used to compute multiplication of two variable bit numbers. The algorithm was reported to solely depend on the ratio of the number of 1's and 0's used to represent a number in binary, rather than on the magnitude of the number. It was mentioned that as the ratio approaches 1, the number of operations required for the multiplication increases and decreases as the ratio tends to move close to 0.

Ramachandran.S, et al, 2012, have thought of an Integrated Vedic multiplier architecture, which by itself selects the appropriate multiplication sutra (UT or Nikhilam) based on the inputs. So depending on inputs, whichever sutra is faster, that sutra is to be selected by the proposed integrated Vedic multiplier architecture. It was implemented for 16 bits but there has not been a clear report on the results or the design.

Kabiraj Sethi, et al, 2012, have proposed a high speed squaring circuit for binary numbers is proposed. High speed Vedic multiplier is used for design of the proposed squaring circuit. Only one Vedic multiplier is used instead of four multipliers as reported previously. In addition, one squaring circuit is used twice.

In paper presented by *G.Ganesh Kumar, et al, 2012*, the Verilog HDL coding of Urdhva tiryakbhyam Sutra for 32×32 bits multiplication and their FPGA implementation by Xilinx Synthesis Tool on Spartan 3E kit have been done and the output has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for calculating the product of 32×32 bits is **31.526 ns**.

The designs of 16×16 bits, 32×32 bits and 64×64 bits Vedic multiplier have been implemented as reported by *Vinay Kumar, 2009* on Spartan XC3S500-5-FG320 and

XC3S1600-5-FG484 device according to this thesis. The computation delay for 16×16 bits Booth multiplier was **20.09 ns** and for 16×16 bits Vedic multiplier was **6.960 ns**. Also computation delays for 32×32 bits and 64×64 bits Vedic multiplier was obtained **7.784 ns** and **10.241 ns** respectively.

A new reduced-bit multiplication algorithm based on Vedic mathematics has been proposed by *Honey Durga Tiwari, et al, 2008.* The framework of the proposed algorithm is taken from Nikhilam Sutra and is further optimized by use of some general arithmetic operations such as expansion and bit shifting to take full advantage of bit-reduction in multiplication. The computational efficiency of the algorithm has been illustrated by reducing a general 44 multiplication to a single 22 multiplication operation.

Manoranjan Pradhan, et al, 2011, have presented the concepts behind the "Urdhva Tiryagbhyam Sutra" and "Nikhilam Sutra" multiplication techniques in their paper. It then shows the architecture for a 1616 Vedic multiplier module using Urdhva Tiryagbhyam Sutra. The paper then extends multiplication to 1616 Vedic multiplier using "Nikhilam Sutra" technique. The 1616 Vedic multiplier module using Urdhva Tiryagbhyam Sutra uses four 88 Vedic multiplier modules; one $16-bit$ carry save adders, and two $17-bit$ full adder stages. The carry save adder in the multiplier architecture increases the speed of addition of partial products. The 1616 Vedic multiplier is reported to have been coded in VHDL, synthesized and simulated using Xilinx ISE 10.1 software. This multiplier is implemented on Spartan 2 FPGA device XC2S30-5pq208.

An integer multiplication algorithm was proposed by Shri Prakesh Dwivedi, 2013, using Nikhilam method of Vedic mathematics which can be used to multiply two binary numbers efficiently taking advantage of the fact that this sutra can convert large-digit multiplication to corresponding small digit multiplication.

Himanshu Thapliyal, et al, 2009, have proposed parallel architectures for computing square and cube of a given number based on Vedic mathematics. For the Xilinx FPGA family, it is observed that for$8-bit$, the gate delay of the proposed square architecture is **28 ns** with area of 90(device utilized) while it is **70 ns** for previously reported squares with area of **77**. For the same operand size, the gate delay in the proposed cube architecture is **28 ns** with area of **90** while for the cube previously reported is **79 ns** with area of **768**. As the operand width is increased to 16, the gate delay of the proposed square architecture increases slightly to **38 ns** with area of **348**(device utilized) while for the square proposed earlier, it significantly increases to **70 ns** with area of **441**. For the operand size of **16**, the cube statistics are found to be **54 ns** with area of **1336** for the proposed Vedic cube while it is **186 ns** with area of 6550 for the cube proposed before.

Chapter 3

Theoretical Background – Algorithms

3.1 Urdhva Tiryakbhyam

This multiplication scheme is best understood by using an example. To illustrate, consider the multiplication of two decimal numbers 325×738. As shown in figure 3.1, the digits on either side of a line are multiplied and the products from each line are added along with the carry from the previous step. This generates one bit of the result as well as a carry. This carry is added in the next step and the process goes on. In each step, the least significant bit (LSB) acts as the result bit and all other bits act as carry for the next step. Initially, the carry is taken to be zero. (*Vinay Kumar, 2009*)

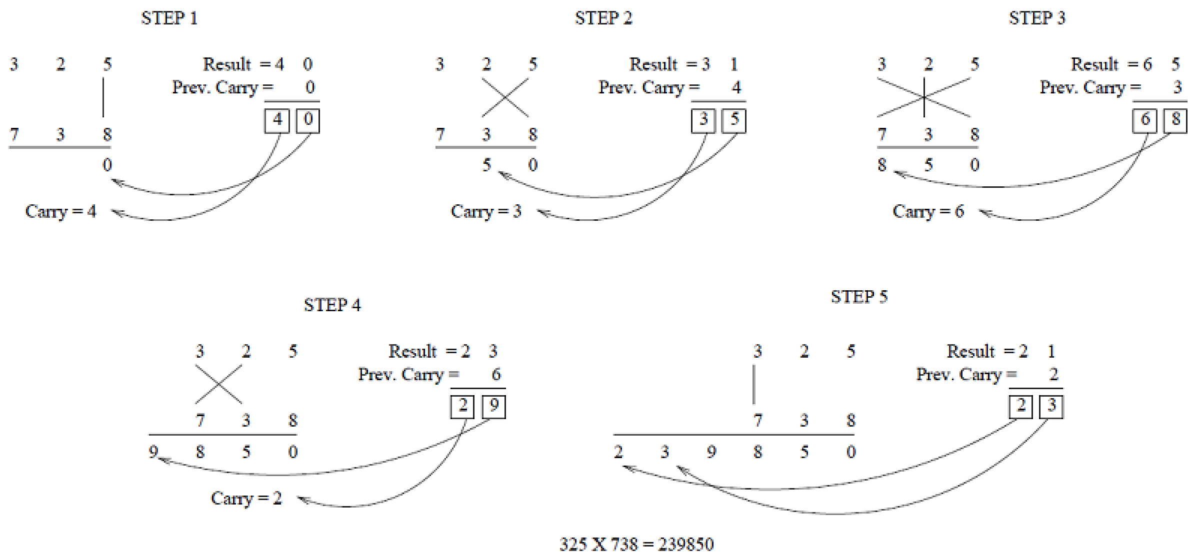

Figure 3.1: UT for a 3×3 Decimal Multiplication

The UT algorithm is based on a novel concept through which the generation of all partial products can be done with the ***concurrent*** addition of these partial products. The algorithm can be easily generalized for $n \times n$ bit multiplication due to its highly modular structure. Since the partial products and their sums are calculated in parallel, the multiplier is ***independent of the clock frequency*** of the processor in case of a synchronous design. Thus the multiplier will require the less amount of time to calculate the product. The net advantage is that it reduces the need of microprocessors to operate at increasingly high clock frequencies. (*Vinay Kumar, 2009*)

While a higher clock frequency generally results in increased processing power, its disadvantage is that it also increases power dissipation which results in higher device operating temperatures. By adopting the Vedic multiplier, microprocessor designers can easily circumvent these problems to avoid catastrophic device failures. The processing power of multiplier can easily be increased by increasing the input and output data bus widths since it has a quite a regular structure. Due to this, layout can be made on a silicon chip easily. The multiplier has the advantage that as the number of bits increases, gate delay and area increases very slowly as compared to other multipliers. Therefore it is time, space and power efficient. (*Vinay Kumar, 2009*)

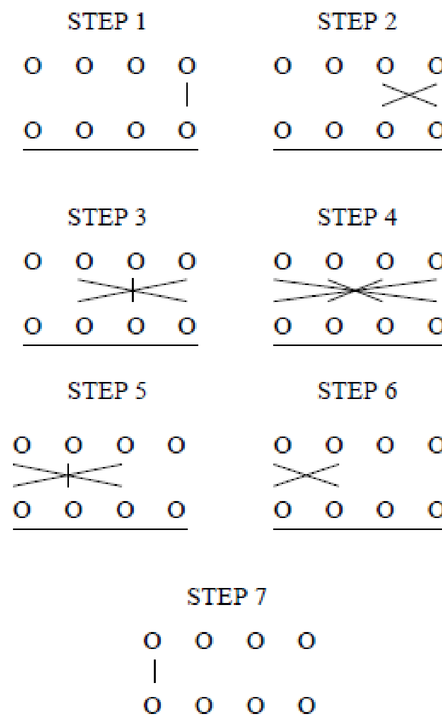

Figure 3.2: UT for a $4 - bit$ multiplication

3.2 Nikhilam

Nikhilam Sutra literally means “All from 9 and last from 10”. Although it is applicable to all cases of multiplication, it is more efficient when the numbers involved are close to the base. It finds the complement of the large number from its nearest base to perform the multiplication operation on it and thus nearer the original number to the base, lesser the complexity of the multiplication. This can be mathematically proven very easily as well. (*Honey Durga Tiwari, et al, 2008*)

Let the two numbers be n and m. Consider a base, b.

$$b - n = p \tag{3.1}$$

$$b - m = q \tag{3.2}$$

$$n \times m = (b - p)(b - q) = b^2 - (p + q)\,b + pq \tag{3.3}$$

$$n \times m = b\,(b - p - q) + pq \tag{3.4}$$

To illustrate, consider the example of multiplying 96×93 in figure 3.3. The base is 100 and hence the complements are 4 and 7 respectively. The product of the complements is given by $4 \times 7 = 28$. The common difference is given by $96 - 7 = 93 - 4 = 89$. This is multiplied with the base and added with the previous product, which turns out to be just a simple concatenation, 8928.

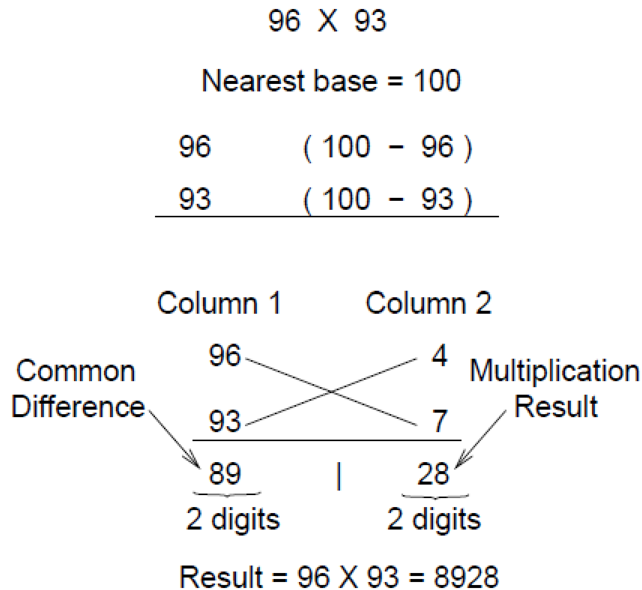

Figure 3.3: Nikhilam Example in Decimal Numbers

3.3 Karatsuba-Ofman Algorithm

The Karatsuba-Ofman algorithm is considered as one of the fastest ways to multiply long integers. It is fundamentally useful in scaling lower bit multipliers to higher bit multipliers. It is based on the 'Divide and Conquer' strategy and has been proved to be asymptotically faster than the standard multiplication algorithm. (*M.Ramalatha, et al, 2009*)

For a $2n-bit$ multiplication, consider X and Y to be the the multiplicand and multiplier respectively. Then, when X_H, X_L, Y_H and Y_L are $n-bit$ numbers, we can write,

$$X = 2^n X_H + X_L \tag{3.5}$$

$$Y = 2^n Y_H + Y_L \tag{3.6}$$

Thus the product of X and Y can be computed as follows,

$$P = X.Y = (2^n X_H + X_L).(2^n Y_H + Y_L) \tag{3.7}$$

Or,

$$P = 2^{2n}(X_H.Y_H) + 2^n(X_H.Y_L + X_L.Y_H) + X_L.Y_L \tag{3.8}$$

It can be observed that

$$X_H.Y_L + X_L.Y_H = (X_H + X_L)(Y_H + Y_L) - X_H.Y_H - X_L.Y_L \tag{3.9}$$

i.e.,

$$P = 2^{2n}(X_H.Y_H) + 2^n\{(X_H + X_L)(Y_H + Y_L) - X_H.Y_H - X_L.Y_L\} + X_L.Y_L \tag{3.10}$$

The standard multiplication algorithm requires **four** $n-bit$ multiplications. But this has just **three** $n-bit$ multiplications along with additions and subtractions as compared to the previous four. But since each multiplication causes more delay as compared to an adder or a subtractor, this will hence result in a more optimized multiplier.

The figures 3.4 & 3.5 show the block diagram for the standard scaling multiplication algorithm and the Karatsuba-Ofman Algorithm respectfully.

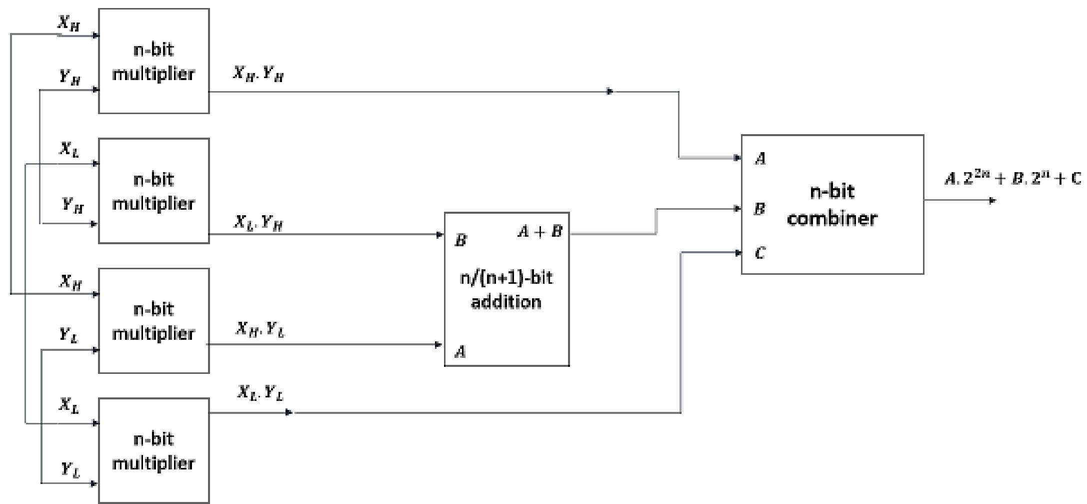

Figure 3.4: Standard Scaling Multiplication Algorithm

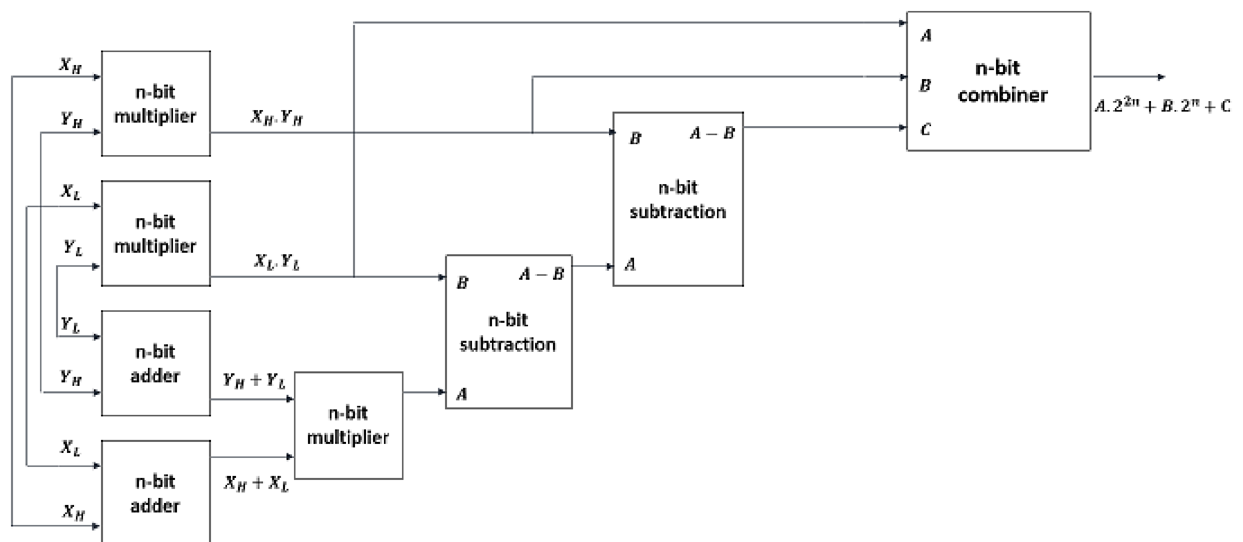

Figure 3.5: Karatsuba-Ofman Algorithm

3.4 Carry Look Ahead Adder

The Carry Look Ahead (CLA) Adder generates carries before the sum is produced using the propagate and generate logic to make addition much faster. Thus the carry chain (the logic that propagates the carry through the full adders of the RCA) is separated from the sum logic (the part of the full adders that produce the sum).

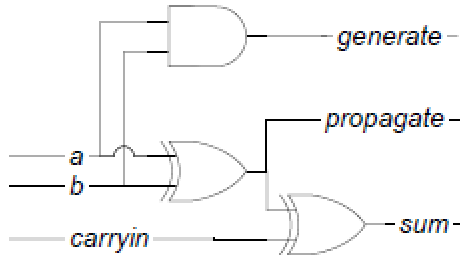

Figure 3.6: Partial Full Adder (PFA)

Consider a $4-bit$ CLA. There are two additional variables called the Generate (G) and Propagate (P) which are fundamental for the CLA logic.

Let A and B be the two $4-bit$ input variables.

$$G_i = A_i.B_i \tag{3.11}$$

$$P_i = A_i \otimes B_i \tag{3.12}$$

$$C_1 = G_0 + P_0.C_0 \tag{3.13}$$

$$C_2 = G_1 + P_1.C_1 = G_1 + P_1.G_0 + P_1.P_0.C_0 \tag{3.14}$$

$$C_3 = G_2 + P_2.C_2 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0 \tag{3.15}$$

$$C_4 = G_3 + P_3.C_3 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.C_0 \tag{3.16}$$

$$S_i = P_i \otimes C_i \tag{3.17}$$

$$GG = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 \tag{3.18}$$

$$PG = P_3.P_2.P_1.P_0 \tag{3.19}$$

These equations show that every carryout in the adder can be determined with just the input operands and initial carryin (C_0). The size of a CLA adder block is chosen as 4 bits. An $8-bit$ CLA can be built from two $4-bit$ CLA blocks, a $16-bit$ CLA from four while a $32-bit$ CLA can be built from two $16-bit$ CLA blocks and so on with the help of the Group Generate (GG) and Group Propagate (PG) pins.

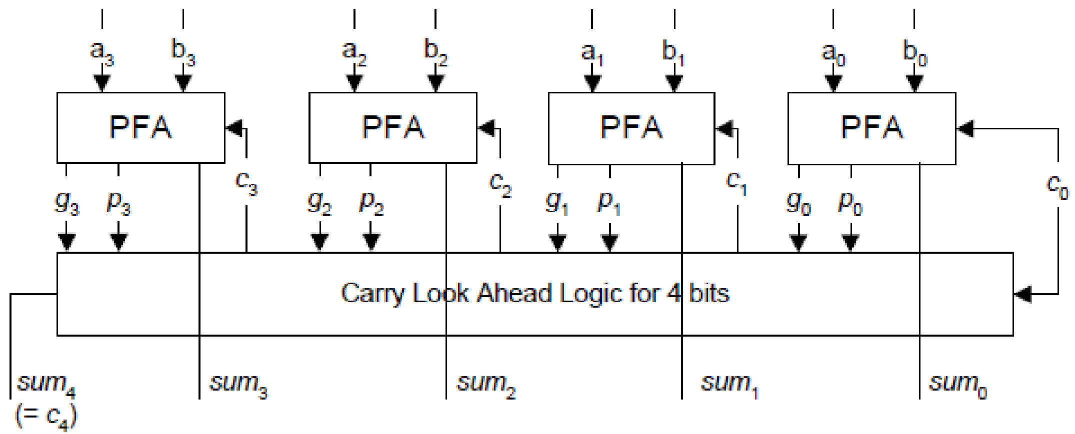

Figure 3.7: $4 - bit$ CLA

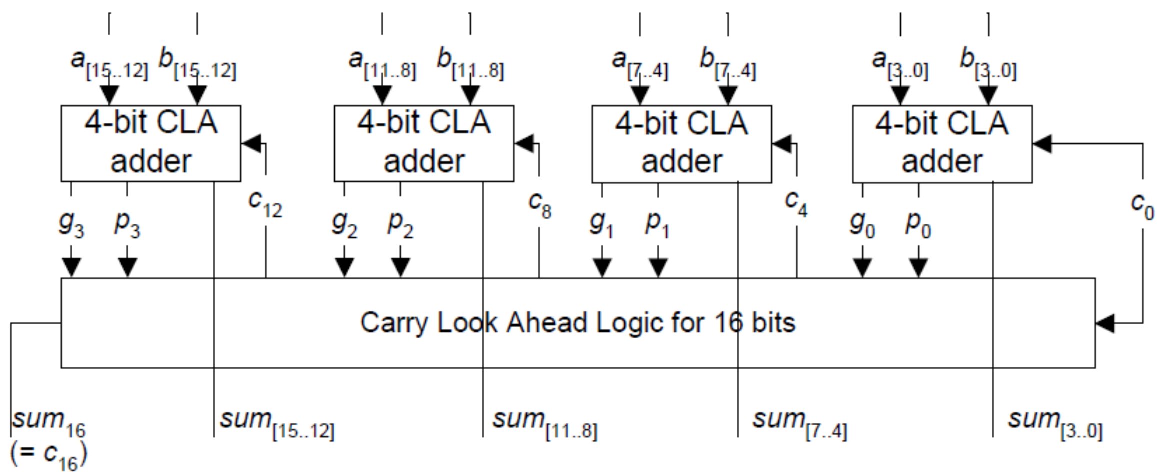

Figure 3.8: $16 - bit$ CLA adder from $4 - bit$ CLA adders

Critical Path Determination

Assuming that all gate delays are the same, the delay for a $4-bit$ CLA adder includes one gate delay to calculate the propagate and generate signals, two gate delays to calculate carry signals, and one gate delay to calculate the sum signals; i.e four gate delays. (*Michael Andrew Lai, 2002*)

For a $16-bit$ CLA adder there is one gate delay to calculate the propagate and generate signal (from the PFA), two gate delays to calculate the group propagate and generate in the first level of carry logic, two gate delays for the carryout signals in the second level of carry logic, and one gate delay for the sum signals. The second level of carry logic for the $16-bit$ CLA adder contributes an additional two gate delays over the $4-bit$ CLA adder, thus increasing the total to six gate delays. Hence,

$$CLA\ levels\ (groupsize = 4) = log_4 N \tag{3.20}$$

$$CLA\ levels\ (groupsize = k) = log_k N \tag{3.21}$$

$$CLA\ gate\ delay = 2 + 2.log_4 N \tag{3.22}$$

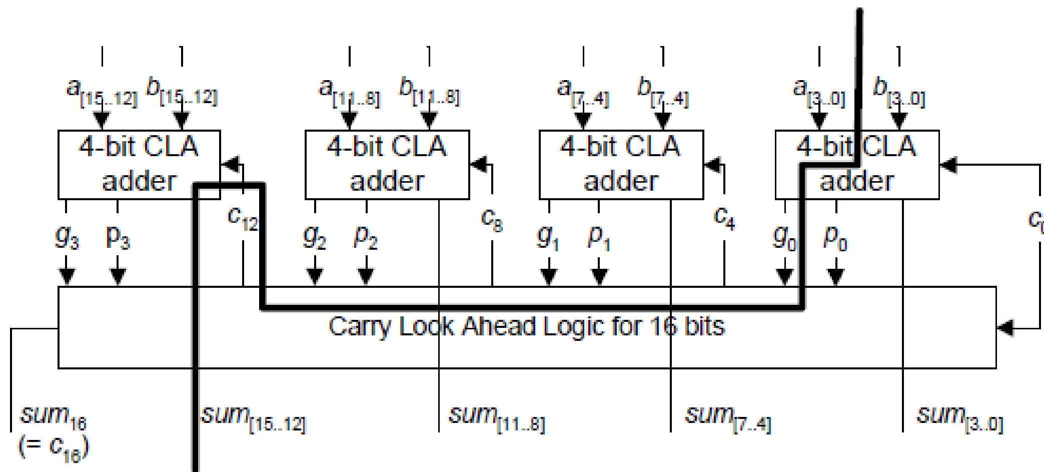

Figure 3.9: Critical Path of a $16-bit$ CLA

In table 3.1, the delay of a CLA adder is **logarithmically dependent** on the size of the adder which theoretically results in one of the fastest adder architectures. And it has the regularity that will allow size adjustment of the adder without much additional design time. It is for these reasons that the CLA architecture is chosen as the adder after comparison with the Ripple Carry Adder (RCA), the Carry Skip Adder (CSKA) and the Linear Carry Select Adder (CSLA). Power consumption of the CLA might be slightly higher as compared to Kogge-Stone Adders, etc but the ease of scaling higher bit adders, the lower area and the lesser delay make this trade-off very slight. (*Michael Andrew Lai, 2002*)

Table 3.1: Adder Comparison – N: input size, k: group size

Adder	Delay	Normalized Area	Normalized Design Time
RCA	N	1	1
CSKA	N/k	1.14	2
CLA	$log_k N$	1.88	8
CSLA	$\frac{log_k N}{(N/k)}$	2.56	10

The Delay column expresses how the delay of the adder is proportional to the length (input size). The next column, Area, normalizes the area for the RCA (based on the subcells) and compares the relative sizes of the other adders to this normalized value. And finally, the Design Time column is an estimate of the normalized time required to design the particular adder based on the RCA design time. (*Michael Andrew Lai, 2002*)

Chapter 4

Design & Implementation

4.1 Synthesizable Code for Hardware Efficiency

The process of automatically converting the description in RTL to gates from the target technology is called ***Synthesis***. It converts the design from a higher level description to a lower one. Normal programming and HDL (Hardware Description Language) programming are usually built over the same platform C. But both are fundamentally different. HDL's (eg. Verilog) are aimed at both simulation and synthesis of digital circuits. All descriptions can be simulated, but only some can be synthesized. And by changing the method of coding, the synthesized design can be made optimum.

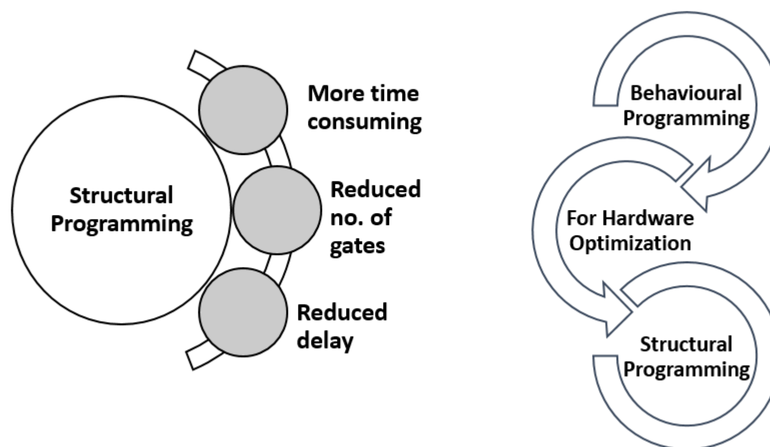

Figure 4.1: Structural Programming

Verilog has two major programming models - **Behavioural** and **Structural**, the former more useful for simulations and the latter for hardware level. However, structural mode is more time consuming and requires more effort. An overall plan is required for implementing structural codes. Developing code as the project progresses will not yield optimal results. But for making the codes truly synthesizable as well as for true hardware optimization, structural programming is a must.

Here, the design of each of the modules along with its implementation using various optimization methods and logical techniques is presented. Also, several new designs and original combinations of existing designs have also been implemented in order to increase the efficiency.

4.2 Vedic UT

The basic unit of the $128 - bit$ multiplier is the $4 - bit$ Vedic multiplier built using the Urdhva Tiryakbhyam logic which was explained for decimal numbers in the previous chapter.

Consider the inputs to be the $4 - bit$ numbers $A[3:0]$ and $B[3:0]$.

$$A[3:0] = a_3a_2a_1a_0 \tag{4.1}$$

$$B[3:0] = b_3b_2b_1b_0 \tag{4.2}$$

Then we have,

$$c_0p_0 = a_0b_0 \tag{4.3}$$

$$c_1p_1 = a_1b_0 + a_0b_1 + c_0 \tag{4.4}$$

$$c_2p_2 = a_2b_0 + a_1b_1 + a_0b_2 + c_1 \tag{4.5}$$

$$c_3p_3 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 + c_2 \tag{4.6}$$

$$c_4p_4 = a_3b_1 + a_2b_2 + a_1b_3 + c_3 \tag{4.7}$$

$$c_5p_5 = a_3b_2 + a_2b_3 + c_4 \tag{4.8}$$

$$c_6p_6 = a_3b_3 + c_5 \tag{4.9}$$

Here, the c_i's can be multi-bit numbers while the p_i's are single-bit numbers. The additions are performed using CLA adders as mentioned before. The code is optimized for synthesis with power and delay in consideration. Higher order UT blocks are built according to the proposed scaling plan mentioned below.

4.3 Scaling – The overall plan

A [2n-1:0]
× B [2n-1:0]
Product[4n-1:0]

AH[n-1:0] AL[n-1:0]
× BH[n-1:0] BL[n-1:0]
Product [4n-1:0]

Figure 4.2: Multiplication of two $2n - bit$ numbers

An $n - bit$ multiplier is used to build a $2n - bit$ multiplier. This uses the Karatsuba-Ofman Algorithm mentioned before, as well as innovative logic and coding techniques for maximum efficiency. After this is implemented, $8 - bit, 16 - bit, 32 - bit, 64 - bit\ and\ 128 - bit$ multipliers can easily be built by substituting $n = 4,\ 8,\ 16,\ 32\ and\ 64$ respectively starting with the basic $4 - bit$ UT multiplier as the building block. The $n - bit$ additions are done using $n - bit$ CLA's.

The scaling up procedure is as follows:

Given:

$\boxed{A[(n-1):0] \times B[(n-1):0] \rightarrow Prod[(2n-1):0]}$ n-bit multiplier

To build:

$\boxed{A[(2n-1):0] \times B[(2n-1):0] \rightarrow Prod[(4n-1):0]}$ 2n-bit multiplier

The input operands are split into higher order and lower order terms as shown.

$$AH\,[(n-1):0] = A\,[(2n-1):n] \quad (4.10)$$
$$AL\,[(n-1):0] = A\,[(n-1):0] \quad (4.11)$$
$$BH\,[(n-1):0] = B\,[(2n-1):n] \quad (4.12)$$
$$BL\,[(n-1):0] = B\,[(n-1):0] \quad (4.13)$$
$$X = AH \times BH \quad (4.14)$$
$$Y = AL \times BL \quad (4.15)$$
$$P = X + Y \quad (4.16)$$
$$Z1 = AH + AL \quad (4.17)$$
$$Z2 = BH + BL \quad (4.18)$$
$$T1 = Z1\,[(n-1):0] \times Z2\,[(n-1):0] \quad (4.19)$$
$$T2 = Z1\,[(n-1):0]\ <<\ n.Z2\,[n] \quad (4.20)$$
$$T3 = Z2\,[(n-1):0]\ <<\ n.Z1\,[n] \quad (4.21)$$
$$TX = T1 + T2 \quad (4.22)$$
$$TY = TX + T3 \quad (4.23)$$
$$x = Z1[n]\ .\ Z2[n] \quad (4.24)$$
$$y = x + TX\,[2n] + TY\,[2n] \quad (4.25)$$
$$Z = \{y, TY\} \quad (4.26)$$
$$R = Z + \overline{P} + 1 \quad (4.27)$$
$$Prod = \{X, Y\} + (R\ <<\ n) \quad (4.28)$$

4.3.1 Where have the optimizations taken place?

1. A more efficient $4-bit$ Vedic Multiplier is developed in structural modelling
2. The faster and efficient CLA adder is used throughout
3. The traditional behavioural left shift operator (<<) is replaced with AND logic
4. $(2n+1)-bit$ addition (which will have to be done by a $4n-bit$ adder) is converted to a $2n-bit$ addition and a single bit addition (Full adder)
5. An efficient combiner logic is developed which requires only one addition and a concatenation

4.4 UT Squaring

It can be seen easily that a special case of multiplication, i.e. Squaring, is much simpler in terms of hardware and software design as compared to the normal situations. This can be exploited to build the integrated multiplier.

For squaring a $4-bit$ number, i.e $A[3:0] = B[3:0]$

$$c_0p_0 = a_0 \tag{4.29}$$
$$c_1p_1 = 2.a_1a_0 + c_0 \tag{4.30}$$
$$c_2p_2 = 2.a_2a_0 + a_1a_1 + c_1 \tag{4.31}$$
$$c_3p_3 = 2.a_3a_0 + 2.a_2a_1 + c_2 \tag{4.32}$$
$$c_4p_4 = 2.a_3a_1 + a_2a_2 + c_3 \tag{4.33}$$
$$c_5p_5 = 2.a_3a_2 + c_4 \tag{4.34}$$
$$c_6p_6 = a_3a_3 + c_5 \tag{4.35}$$

Thus, since multiplication by 2 is just a **left shift by 1 bit**, the square is a very special case that is easy to implement and which consumes less area, power and delay as compared to the normal Vedic multiplier.

4.4.1 Scaling – Adapted for Squares

$A_H[n-1:0]\ A_L[n-1:0]$
$\times\ A_H[n-1:0]\ A_L[n-1:0]$
A-square$[4n-1:0]$

Figure 4.3: $2n-bit$ Squaring Unit

Building a $2n - bit$ squaring unit from an $n - bit$ squaring unit is much easier than building a $2n - bit$ multiplier from an $n - bit$ multiplier.

$$AH\left[(n-1):0\right] = A\left[(2n-1):n\right] \tag{4.36}$$
$$AL\left[(n-1):0\right] = A\left[(n-1):0\right] \tag{4.37}$$

$$X = AH^2 \tag{4.38}$$
$$Y = AL^2 \tag{4.39}$$
$$Z = 2\left(AH \times AL\right) \tag{4.40}$$
$$Prod = \{X, Y\} + (Z \ << \ n) \tag{4.41}$$

As can be seen, the 15 steps in scaling the normal UT multiplier have been reduced to just 4 for scaling the UT square. Again, the multiplication by 2 is just a left shift by 1 bit as mentioned before.

4.5 Proposed Design D

Given two operands A and B, assuming without loss of generality, $A > B$, it can be observed that,

$$\begin{aligned} A \times B &= \frac{4.AB}{4} && (4.42) \\ &= \frac{(A+B)^2 - (A-B)^2}{4} && (4.43) \\ &= \left(\frac{A+B}{2}\right)^2 - \left(\frac{A-B}{2}\right)^2 && (4.44) \\ &= \left(\frac{A+B}{2}\right)^2 - \left(\frac{A+B}{2} - B\right)^2 && (4.45) \\ & && (4.46) \end{aligned}$$

Thus,

$$A \times B = (av)^2 - (dev)^2 \tag{4.47}$$

A multiplication is broken down into the difference between two squares. Since only integers are being dealt with here, this would work only when the operands are both even or both odd. Otherwise the average would turn out to be a floating point.

Squaring is less costly in power expenditure, area occupancy and delay. This design takes advantage of this fact and makes it possible for it to be used for non-square integers too.

4.6 Thresholding Nikhilam

Here the basic idea of Nikhilam is extended to binary. Since Nikhilam is efficient when the inputs are very near to the base, a **threshold** is chosen, upto which we can consider Nikhilam to be better. It is estimated that the most optimum threshold would be $(1/4)^{th}$ the input size.

For a $16 - bit$ multiplier, a possible threshold can be 4 bits. This means that if the difference between the base and number comes out to be a $4 - bit$ integer, then Nikhilam is considered better. For the multiplication of the $4 - bit$ complements, the already optimized $4 - bit$ Vedic Multiplier will be used. Thus, a $16 - bit$ multiplication will be reduced to a $4 - bit$ multiplication along with addition & subtraction.

Table 4.1: Thresholds for various multipliers

Input size	Threshold Value
8 bits	2 bits
16 bits	4 bits
32 bits	8 bits
64 bits	16 bits
128 bits	32 bits

Range of Inputs

Since Nikhilam is efficient for a certain range of inputs, it should be determined if the given inputs belong to that range.

Given the size of the input as is and the threshold size as ts, we can find the range of values that can be used effectively with Nikhilam, $range$, as follows. Base, $B = 2^{(is-1)}$ and threshold value, $Th = 2^{ts} - 1$

$$range \quad \epsilon \quad [B - Th, B + Th] \tag{4.48}$$

Consider the input to be m and n, and the base, b.

Case 1: Both inputs are greater than the base

Table 4.2: Multiplication of $m \times n$, base r

	Integer	Base difference
Multiplicand	m	$m - r = a$
Multiplier	n	$n - r = b$
	$m + n - r$	$a \times b$
Result	$r(m + n - r) + ab$	

Case 2: Both inputs are less than the base

Table 4.3: Multiplication of $m \times n$, base r

	Integer	Base difference
Multiplicand	m	$r - m = a$
Multiplier	n	$r - n = b$
	$r - m - n$	$a \times b$
Result	$r(r - m - n) + ab$	

Case 3: One input (m) is greater than the base & the other (n), less

Table 4.4: Multiplication of $m \times n$, base r

	Integer	Base difference
Multiplicand	m	$m - r = a$
Multiplier	n	$[n - r = b = -q] \rightarrow [r - n = q]$
	$m + n - r$	$a \times (-q)$
Result	$r(m + n - r) - aq$	

Steps to be followed

1. Choose base r for the inputs m and n
2. Decide if the inputs belong to the correct range of inputs for Nikhilam
3. Decide which case of Thresholding Nikhilam to use

Optimization Points

1. Base can always be chosen as $2^{(is-1)}$ for optimum performance.
2. The adders used are CLA adders.
3. A higher order multiplication is reduced to a few addition operations and a lower order multiplication.

4.7 Successive Nikhilam

Instead of applying Nikhilam only for a select few numbers as given by the threshold, we can use it for all integers by applying it successively. In this case, the entire process of multiplication is broken down into addition and subtraction.

Table 4.5: Nikhilam – 1111 × 1111, base = 1000

Bits	Base Difference	Next Difference	Next Difference
$m = 1111$	$1111 - 1000 = 111$	$111 - 100 = 11$	$11 - 10 = 1$
$n = 1111$	$1111 - 1000 = 111$	$111 - 100 = 11$	$11 - 10 = 1$
			$1 \times 1 = 1$
		$10(11 + 1) + 1 = x_1$	
	$100(111 + 11) + x_1 = x_2$		
$1000(1111 + 111) + x_2 = x_3$			
$Ans = 11100001$			

Consider the example in table 4.5 – 1111 × 1111. This has reduced to just one AND ($1 - bit$) operation, additions and shifting (multiplying by the base is basically left-shifting). To account for multiplications which involve different sizes, a little more thought has to be given. The final algorithm designed is as given below.

Given $is - bit$ inputs, m and n

$$p = m \otimes n \tag{4.49}$$
$$q = m \otimes (is)'b0 \tag{4.50}$$
$$S_0 = m[0] \times n[0] \tag{4.51}$$
$$for \longrightarrow i = 1 : (is - 1) \tag{4.52}$$
$$case(p[i], q[i]) \tag{4.53}$$
$$00 \quad \rightarrow \quad S_i = 0 \tag{4.54}$$
$$01 \quad \rightarrow \quad S_i = m[i : 0] + n[(i - 1) : 0] \tag{4.55}$$
$$10 \quad \rightarrow \quad S_i = m[(i - 1) : 0] \tag{4.56}$$
$$11 \quad \rightarrow \quad S_i = n[(i - 1) : 0] \tag{4.57}$$
$$endcase \tag{4.58}$$
$$m \times n = \sum_{i=0}^{(is-1)} S_i \tag{4.59}$$

Optimization Points

1. Since only one $1 - bit$ AND operation is required along with aditions and shiftings, as the input size increases, Successive Nikhilam will grow more efficient.

2. One disadvantage is for an $n - bit$ input, it requires n steps.

3. This is basically recursion. Hence implementing it in Structural Modelling is more difficult and it can prove costly in terms of power and delay.

Since addition is performed using CLA's, for an $n-bit$ multiplier,

Table 4.6: Total No. of Adders Required – Successive Nikhilam

CLA Type	# required
$CLA_{(2n)}$	$n/2$
$CLA_{(n)}$	$n/2$
$CLA_{(n/2)}$	$n/4$
.	.
.	.
$CLA_{(2)}$	1

Because the number of adders can be predetermined for a multiplier based on its input size, it is very easy to give a nearly estimate of the power consumption and the area occupied, given the details of the adders. More than any other multiplier, Successive Nikhilam almost solely depends on the adders since the only multiplication that takes place is a $1-bit$ AND operation.

Chapter 5

Sampoornam – the Proposed Integrated Multiplier

A novel multiplier, which will integrate maximum of the advantages in each design implemented in the above chapter, is proposed. It is designed to have a specialized logic unit that will decide which multiplier is to be used for optimum results of all the given choices, based on the input values. This will be a thorough multiplier with no human intervention required. Since it is meant to be absolute and completely based on Vedic roots, it would be apt to name it as "***Sampoornam***" or the "**Absolute Vedic**" multiplier.

5.1 Sampoornam: Specifications

There are around 7 possible designs to choose from, as proposed in this report, in order to multiply two given integers, apart from the case when one of the inputs is a 0 (**Output 0**).

They are:

1. Vedic multiplier based solely on Urdhva Tiryakbhyam – **Vedic UT**
2. Proposed design D for multiplication derived from squaring logic – **Design D**
3. Nikhilam multiplier with both inputs above the base – **Nikh GG**
4. Nikhilam multiplier with both inputs below the base – **Nikh SS**
5. Nikhilam multiplier with one input above and one below – **Nikh SG**
6. Squaring logic based on Urdhva Tiryakbhyam – **UT Square**
7. When either of the numbers is equal to the base – **Left Shift**

The Successive Nikhilam is not a good candidate for integrating into the ***Sampoornam*** multiplier since it is based on recursion while rest of the blocks are not. And the power consumed, area occupied and the delay will be more than normal if it is to be included. So the only possible options are those listed above. Henceforth in this report, Nikhilam refers to Thresholding Nikhilam unless otherwise specified.

5.2 Sampoornam: Logic

Consider the inputs to be x and y, with the base as b. Let t be the threshold value for Nikhilam.

Table 5.1: Designs and their input constraints

Design	Input Constraints
Left Shift	x or $y = b$
UT Square	$x = y$
Nikh SG	$x \epsilon (b, b+t)$ and $y \epsilon (b-t, t)$
Nikh GG	$x, y \epsilon (b, b+t)$
Nikh SS	$x, y \epsilon (b-t, b)$
Design D	$x - y = 2k$, k is an integer
Output 0	x or $y = 0$

However, on performing the analyses, as will be seen in the Chapter 6, it is found that Design D implemented with CLA Adder comes to be less optimum than the Vedic UT multiplier. For this reason, the Design D is not included in the final integrated multiplier – ***Sampoornam***. Other inferences about this will be pointed out in Chapter 7.

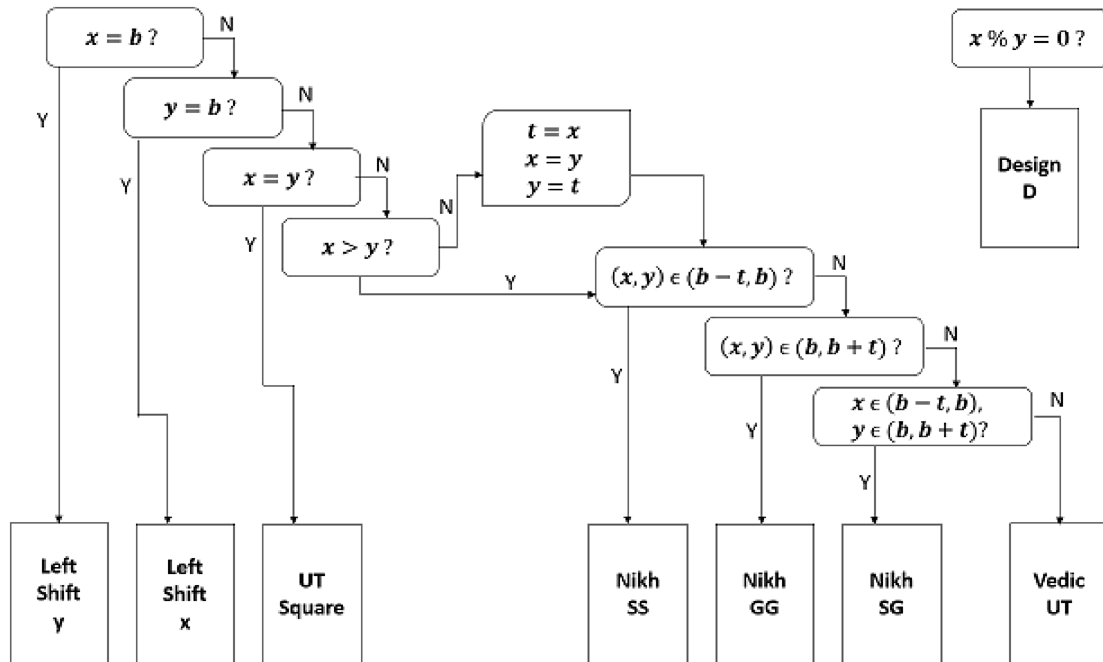

Figure 5.1: Sampoornam Logic – Flowchart

5.3 Sampoornam: Implementation

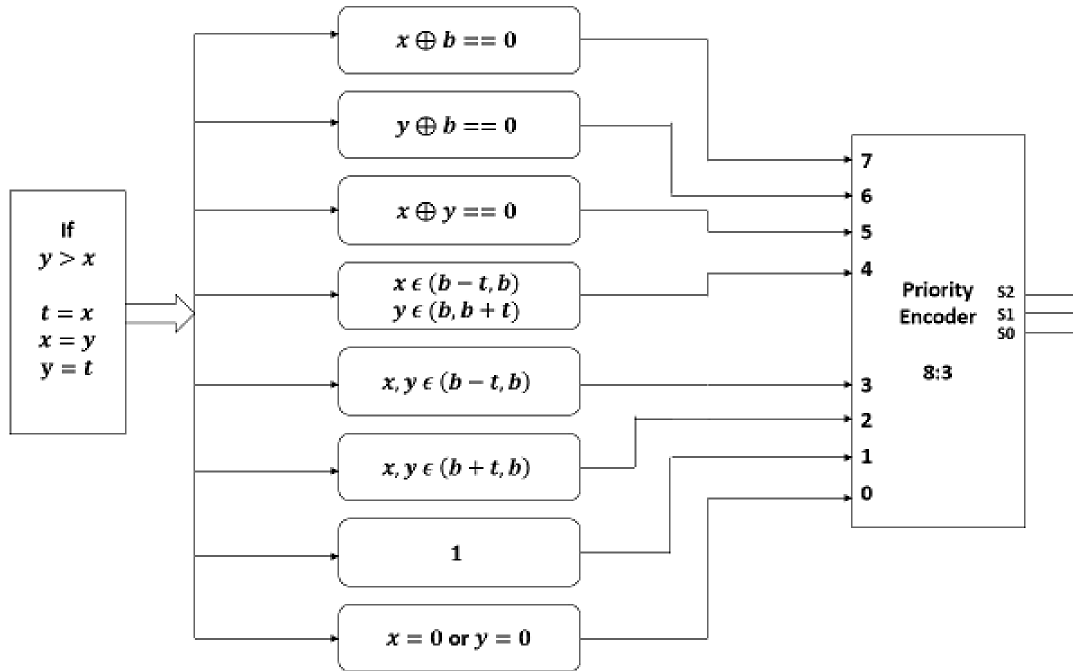

Figure 5.2: Sampoornam – Implementation

The code is implemented according to the algorithm given above. A point to be noted is that, this is implemented completely using Combinational Logic Gates along with comparators. This could have been very simply written as a set of ***if*** conditions, which would give the same output. However, then the logic block will not be optimum and will tend to draw more power, as well as occupy more area. This is so designed in order to be in accordance with Structural Programming.

Table 5.2: Priority Encoder – Output

$S_2S_1S_0$	Design Selected
111	Left Shift y
110	Left Shift x
101	UT Square
100	Nikh SG
011	Nikh SS
010	Nikh GG
001	Vedic UT
000	Output 0

5.4 Sampoornam: Advantages

1. Only one of the designed modules will be ON at any given time. Hence, the power drawn overall will be split among them.
2. The Logic block (which decides which module will be ON) is completely separated from the modules. That is, the modules and the logic block can be independently optimized.
3. Since some modules are very highly optimum for a given range of inputs, using them only for those inputs is going to increase the overall efficiency of the multiplier.
4. This multiplier is easily hardware realizable as well, having proper modularity and structure. Multiplexers can be employed instead of the Priority Encoder in certain situations.
5. No manual intervention is required during any step for implementing the logic.

It is hoped that the ***Sampoorna Multiplier*** will be truly complete and absolute. The results of various analyses that was done on it is given in the following chapters.

Chapter 6

Simulations and Results

The code is written in Verilog HDL using the Xilinx ISE 14.3 Design Suite. Test benches are written to validate the codes. Then the codes are fed to the Cadence Encounter RC Compiler, i.e RTL Compiler and the parameters and paths are specified to perform the power analysis, to determine the area occupancy and the worst path delay for each module.

6.1 Vedic UT

6.1.1 RTL Schematics

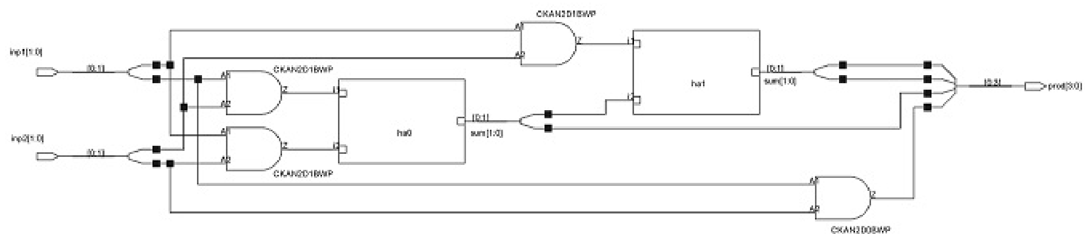

Figure 6.1: $2-bit$ Vedic UT

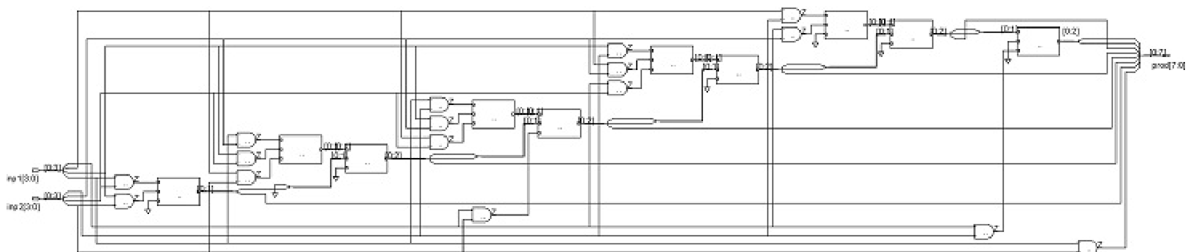

Figure 6.2: $4-bit$ Vedic UT

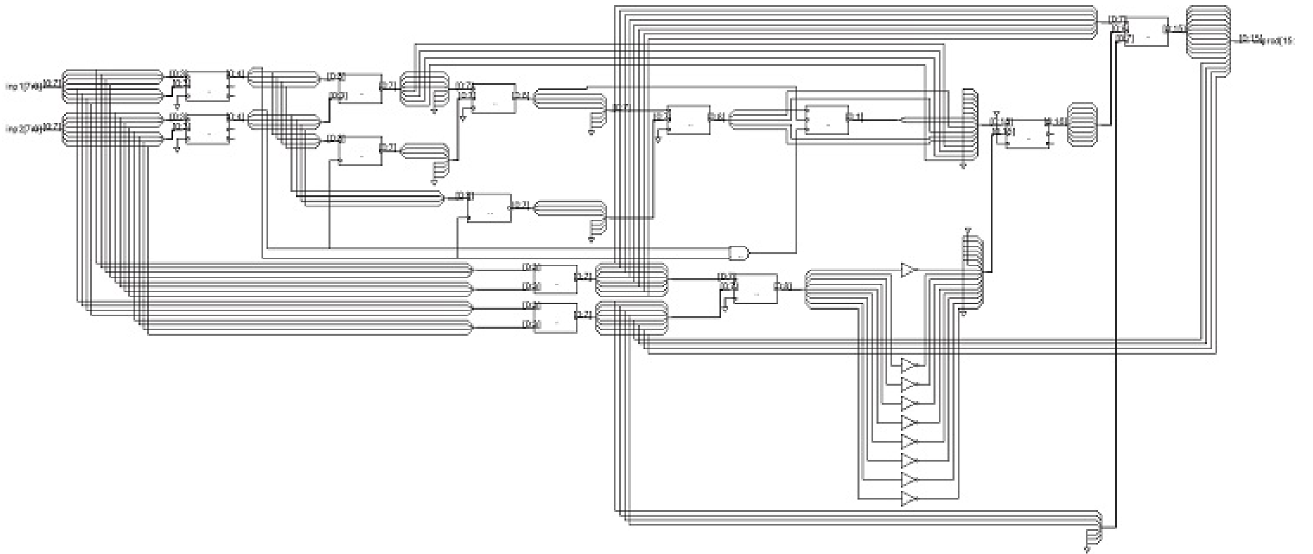

Figure 6.3: $8-bit$ Vedic UT

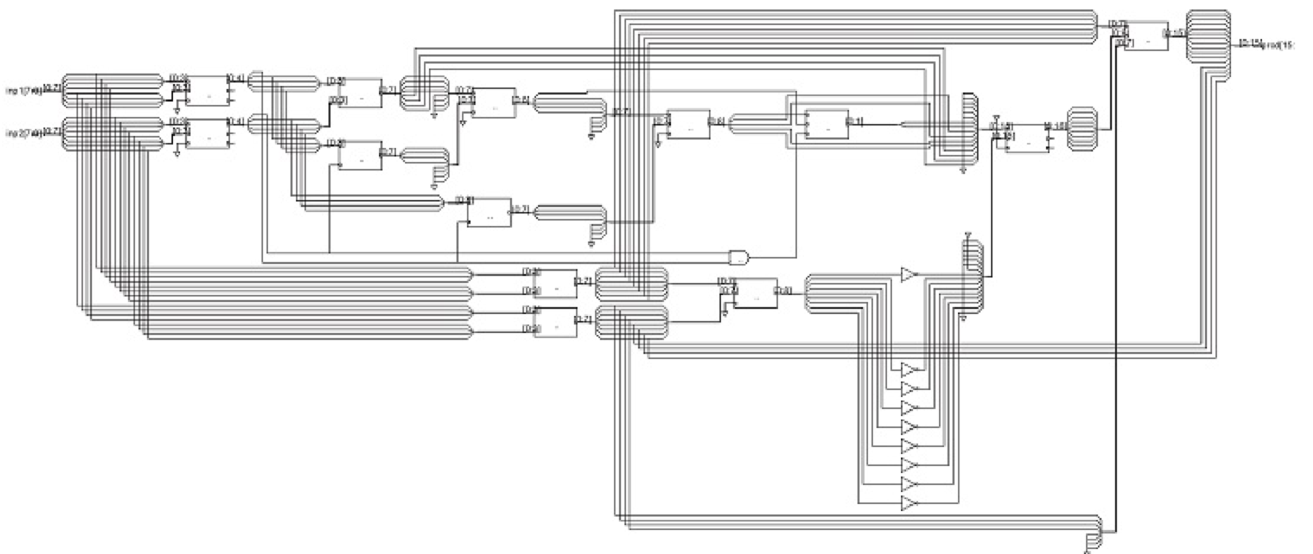

Figure 6.4: $16-bit$ Vedic UT

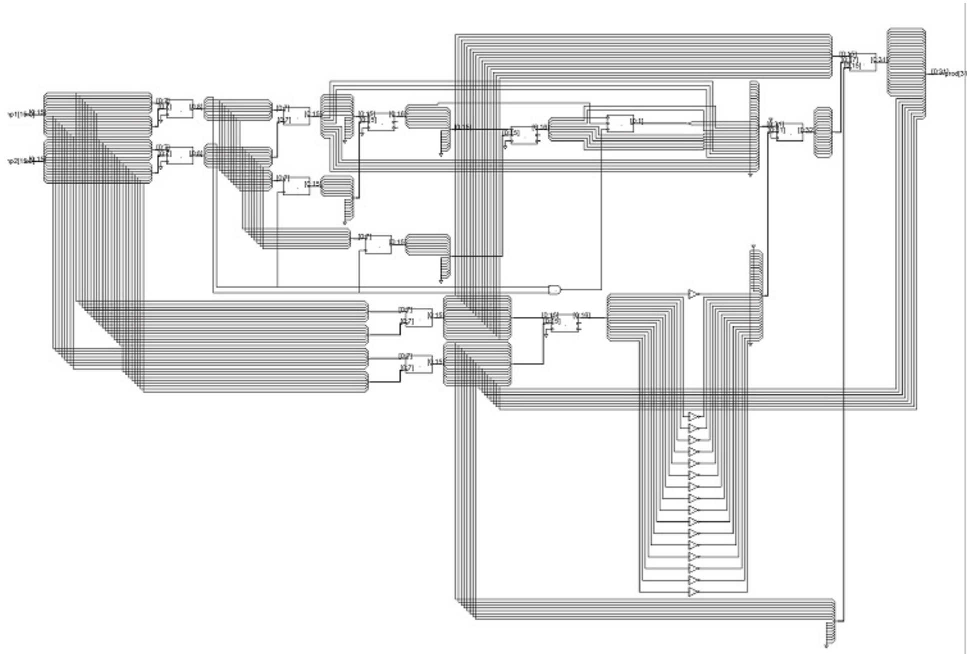

Figure 6.5: $32 - bit$ Vedic UT

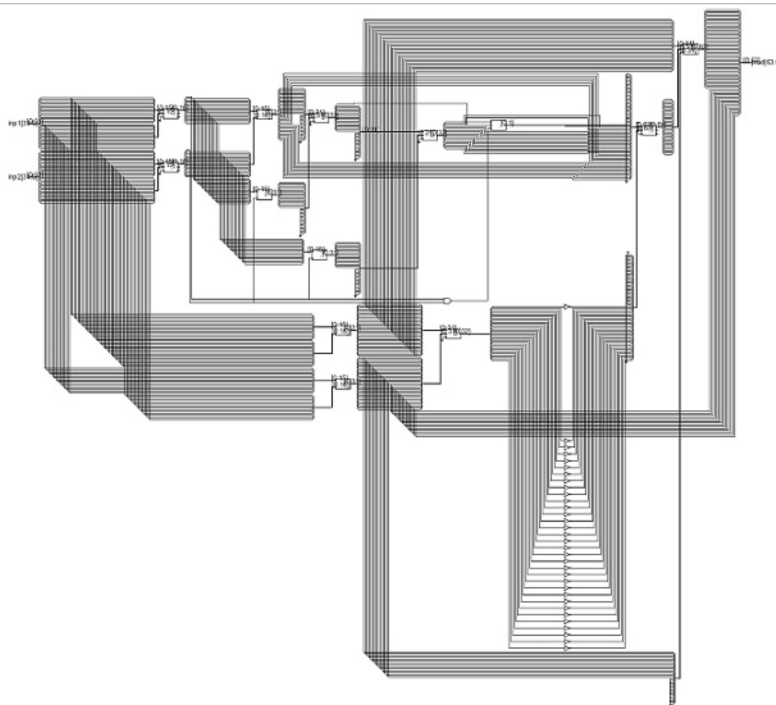

Figure 6.6: $64 - bit$ Vedic UT

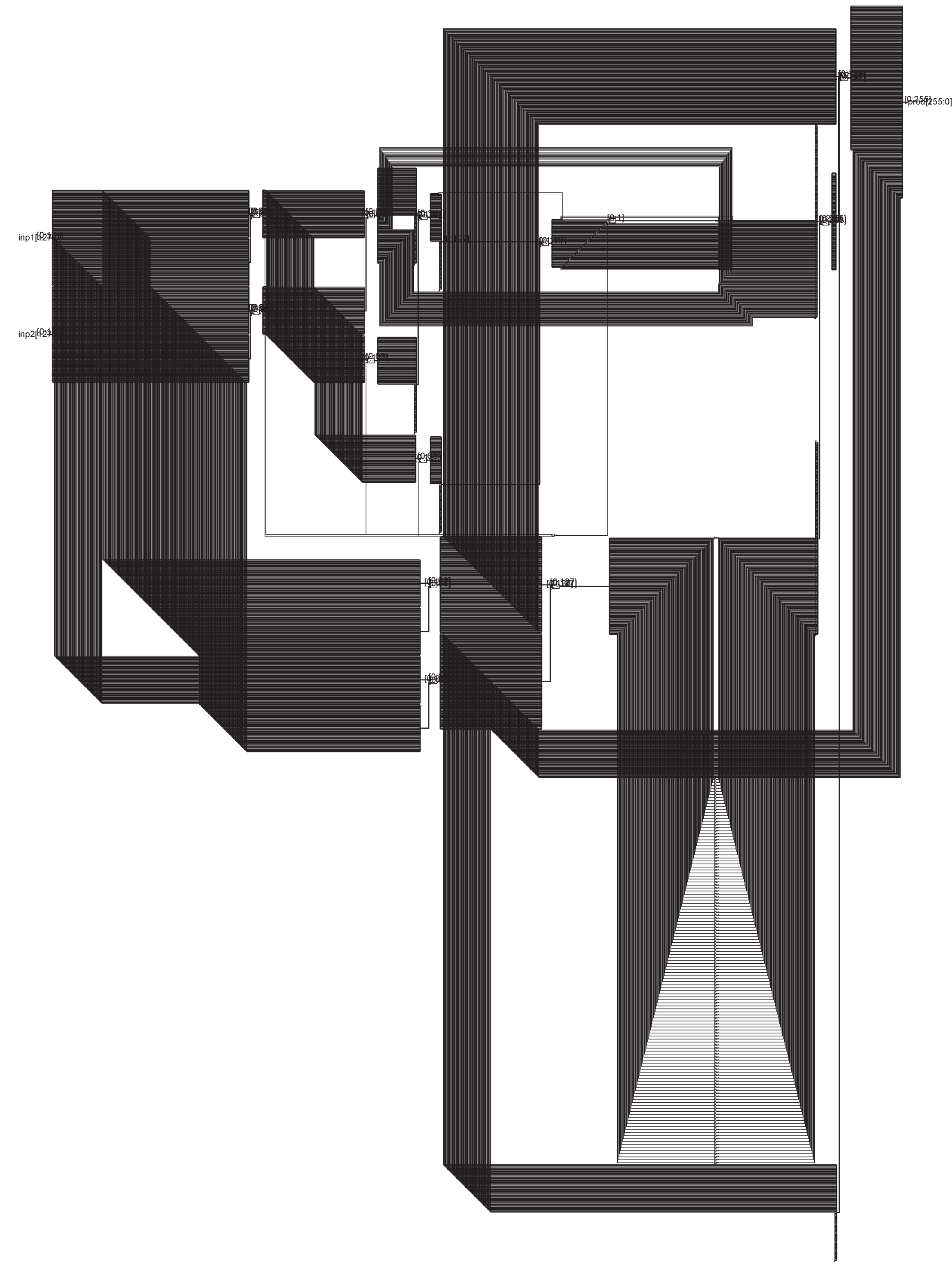

Figure 6.7: $128 - bit$ Vedic UT

6.1.2 RTL Results

Table 6.1: Vedic UT

# bits	Area(nm^2)	Power(nW)	Delay(ps)
2	10	954.171	62.8
4	71	7782.601	340
8	454	63223.888	814.90
16	1913	360307.407	1500.20
32	6909	1688282.769	2321.50
64	23346	7212137.495	3522.10
128	75013	28739302.317	4970.80

6.2 Vedic Square

6.2.1 RTL Schematics

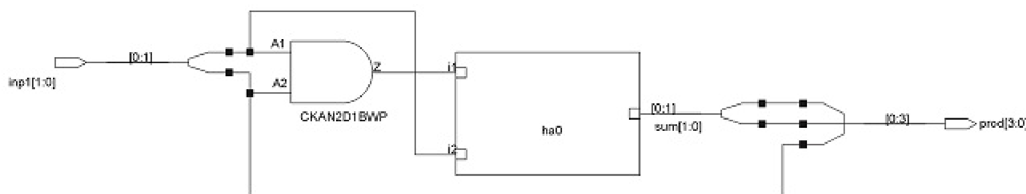

Figure 6.8: $2-bit$ Vedic Square

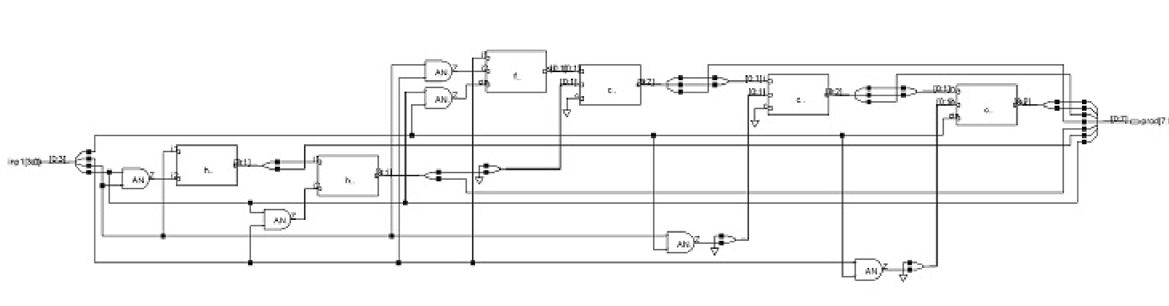

Figure 6.9: $4-bit$ Vedic Square

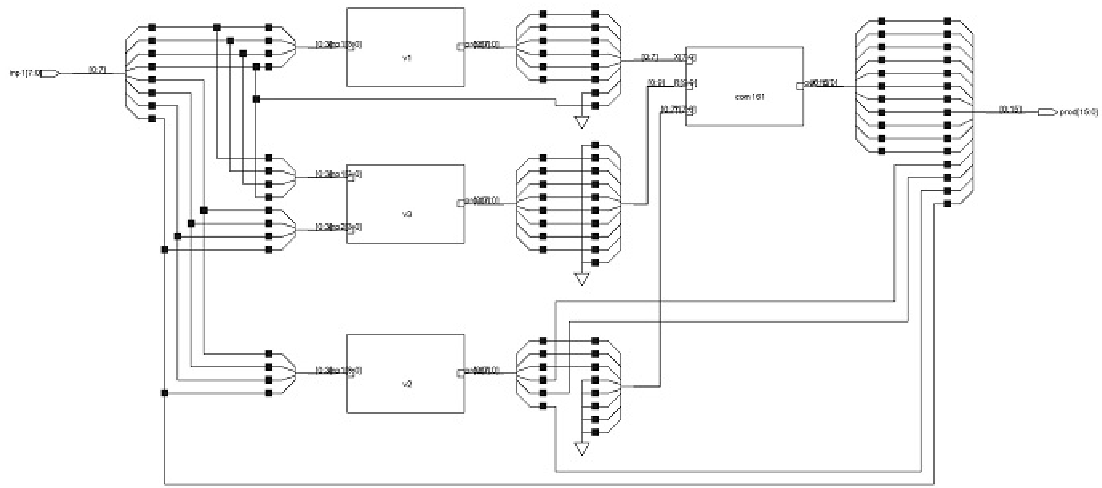

Figure 6.10: $8 - bit$ Vedic Square

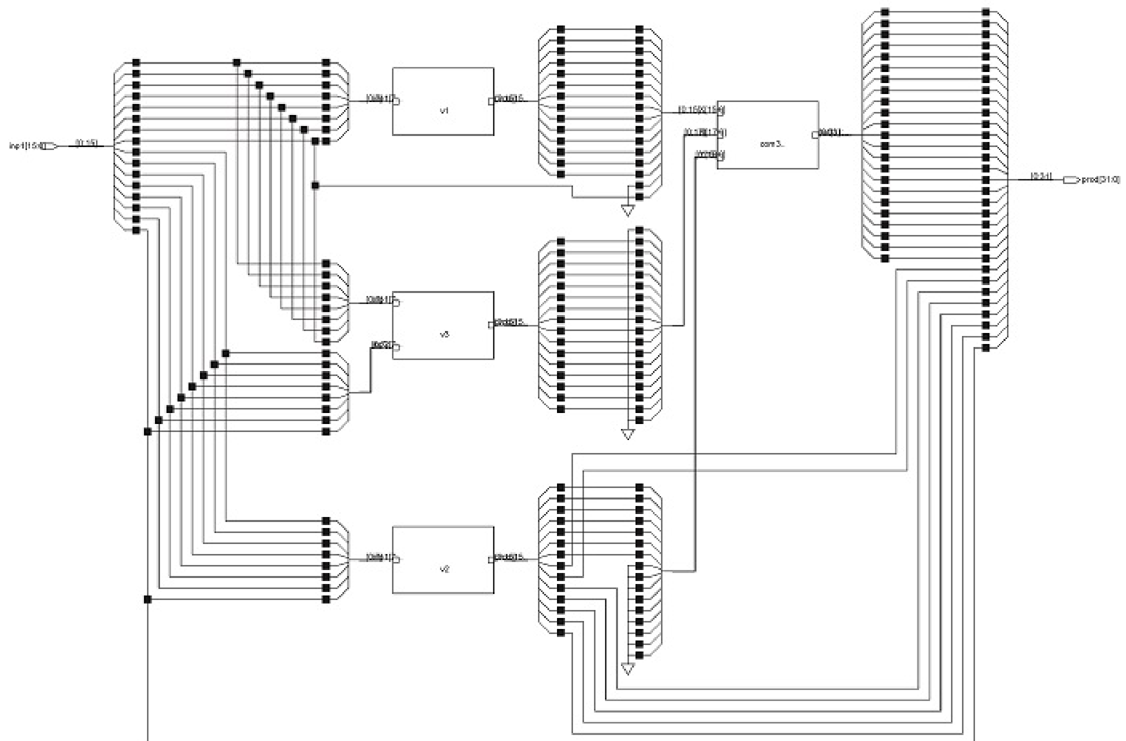

Figure 6.11: $16 - bit$ Vedic Square

6.2.2 RTL Results

Table 6.2: Vedic Square

# bits	Area(nm^2)	Power(nW)	Delay(ps)
2	4	378.573	39.2
4	35	3285.703	240.3
8	187	19757.673	518
16	943	119473.143	1029.10
32	4049	663927.624	1791
64	15542	3231041.100	2615.60
128	55720	14399463.453	3933.2

6.3 Design D

6.3.1 RTL Results

Table 6.3: Design D

# bits	Area(nm^2)	Power(nW)	Delay(ps)
8	527	77934.629	787.20
16	2259	445024.996	1446.80
32	8944	23973.226	2343.40
64	32880	109168939.094	3336.30
128	115260	47426654.289	4811.20

6.4 Nikhilam GG

6.4.1 RTL Schematics

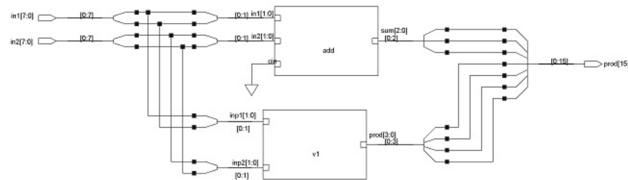

Figure 6.12: $8 - bit$ Nikh GG

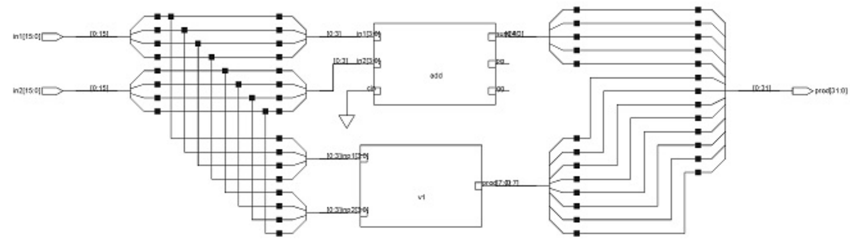

Figure 6.13: $16 - bit$ Nikh GG

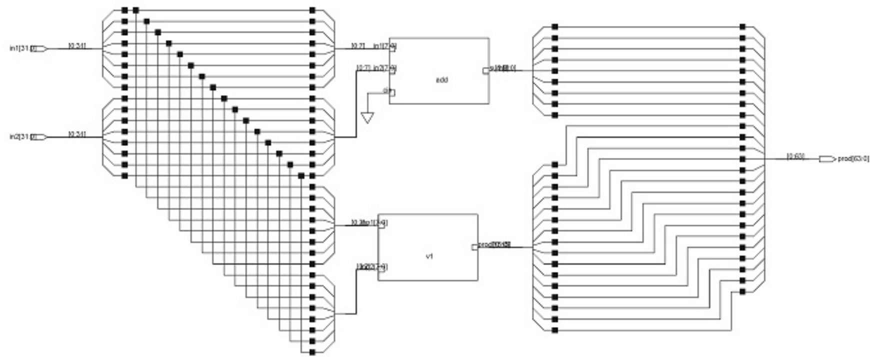

Figure 6.14: $32 - bit$ Nikh GG

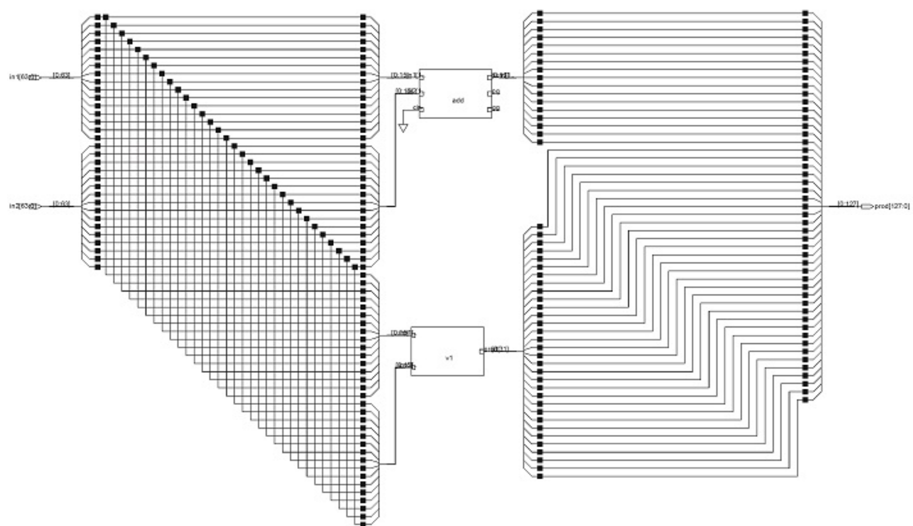

Figure 6.15: $64 - bit$ Nikh GG

6.4.2 RTL Results

Table 6.4: Nikh GG

# bits	Area(nm^2)	Power(nW)	Delay(ps)
8	18	1663.124	62.8
16	87	9331.961	340
32	497	68958.150	814.90
64	2011	370320.247	1500.2
128	7121	1738297.467	2326.70

6.5 Nikhilam - SG

6.5.1 RTL Schematics

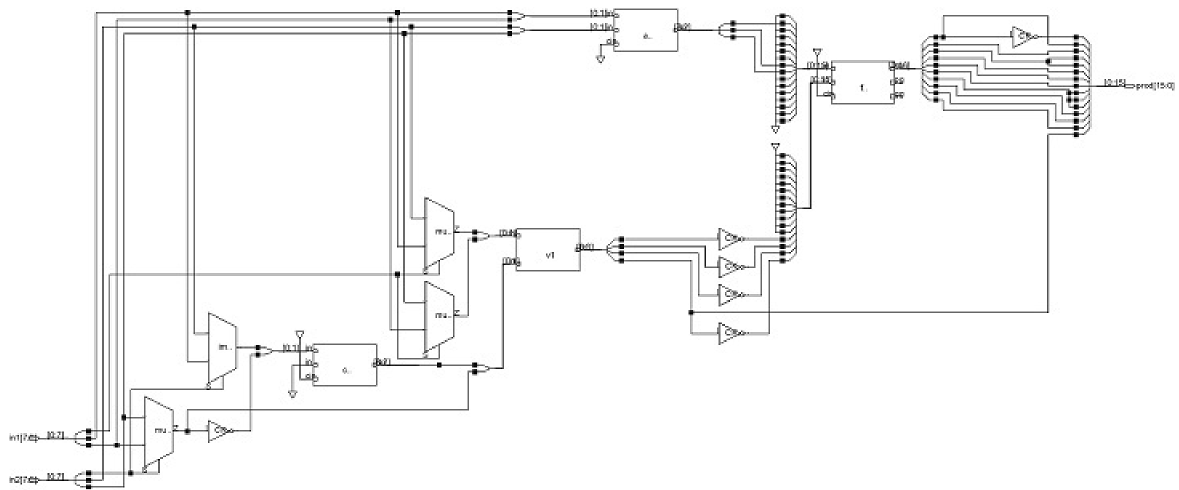

Figure 6.16: $8 - bit$ Nikh SG

.

.

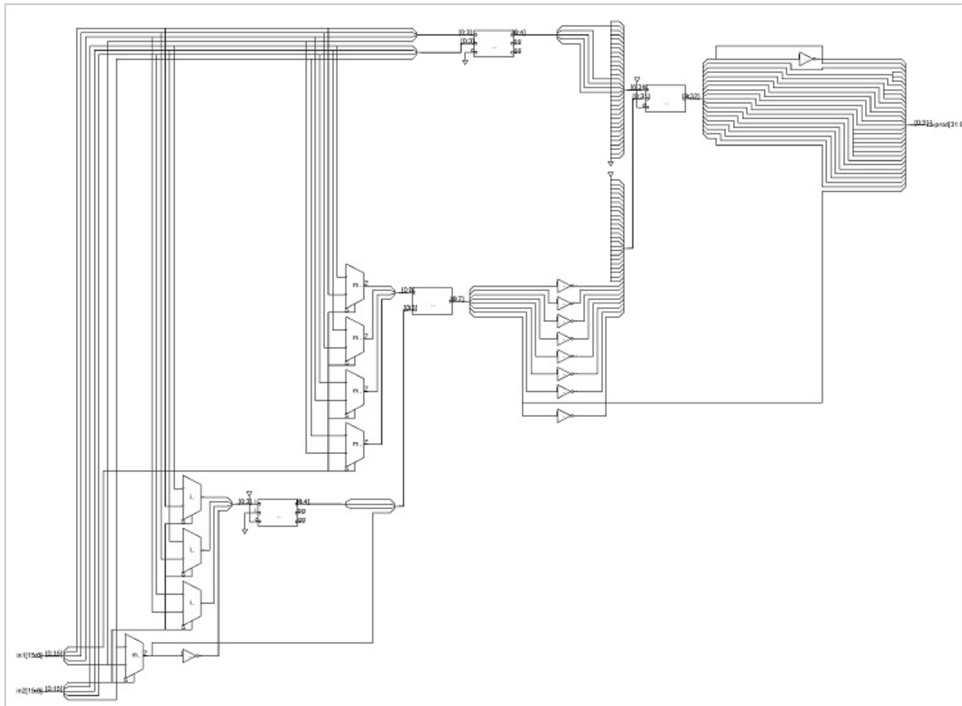

Figure 6.17: $16 - bit$ Nikh SG

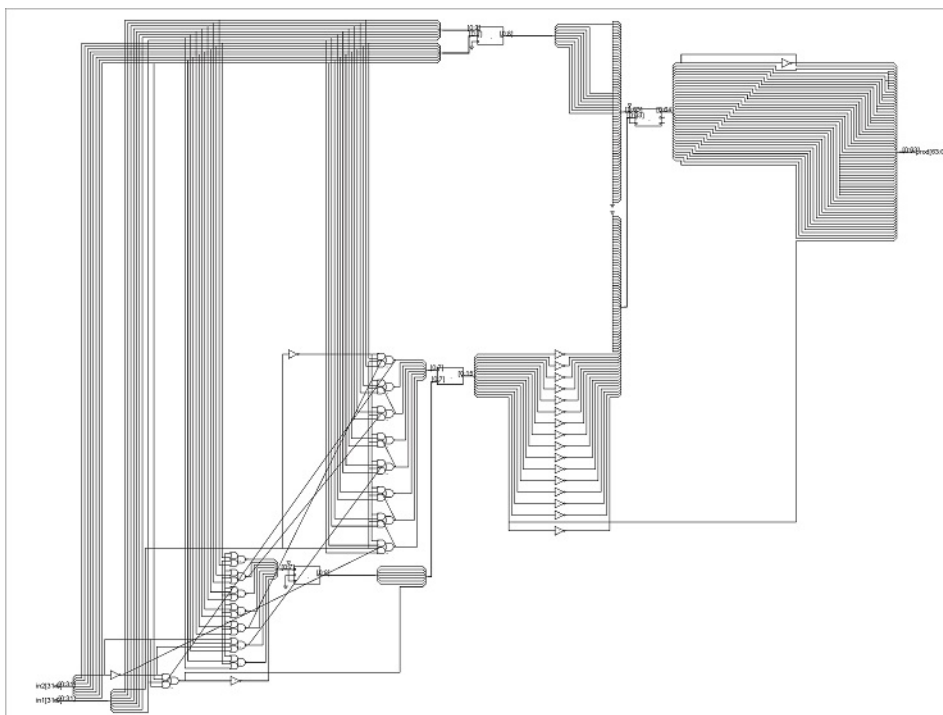

Figure 6.18: $32 - bit$ Nikh SG

6.5.2 RTL Results

Table 6.5: Nikh SG

# bits	Area(nm^2)	Power(nW)	Delay(ps)
8	52	4700.96	262.40
16	155	16786.435	590.40
32	635	80516.342	1185.00
64	2276	396173.188	1968.90
128	7649	1706659.996	2915.60

6.6 Nikhilam SS

6.6.1 RTL Schematics

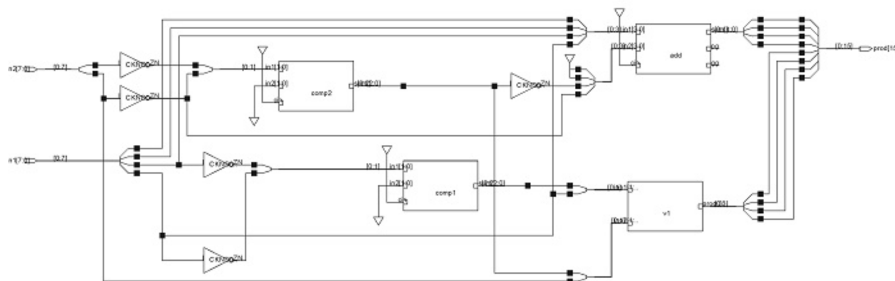

Figure 6.19: $8 - bit$ Nikh SS

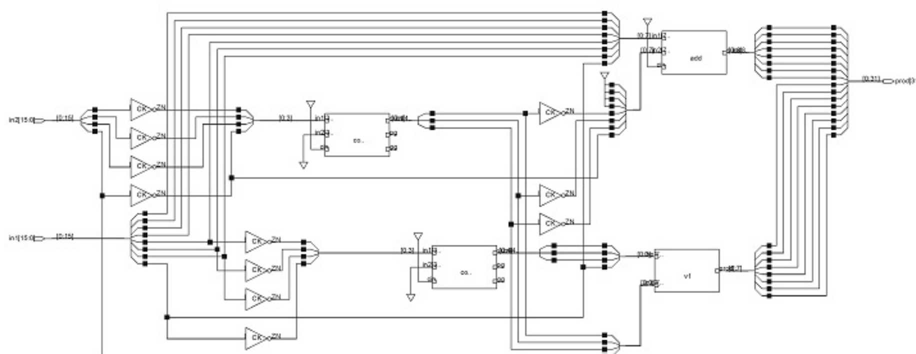

Figure 6.20: $16 - bit$ Nikh SS

.

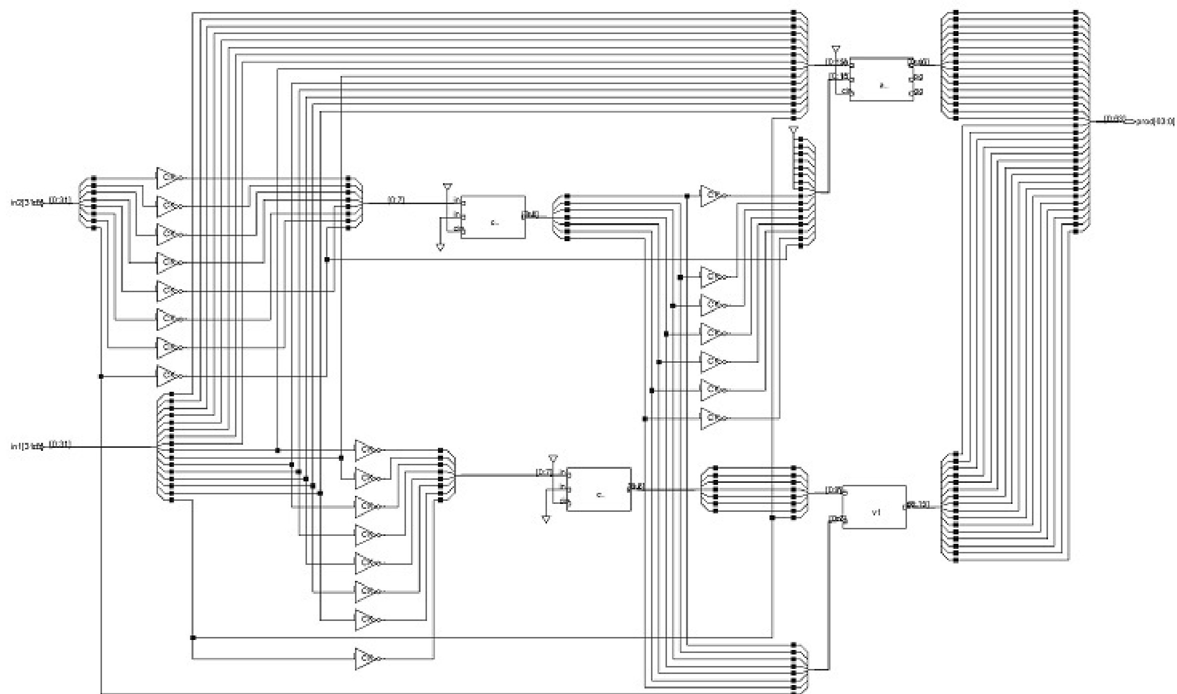

Figure 6.21: $32 - bit$ Nikh SS

.

.

.

6.6.2 RTL Results

Table 6.6: Nikh SS

# bits	Area(nm^2)	Power(nW)	Delay(ps)
8	28	2739.485	144.10
16	122	13466.568	408
32	575	81261.636	960.10
64	2174	379401.386	1691.20
128	7452	1760731.869	2577.60

.

.

6.7 Logic Block

6.7.1 RTL Schematics

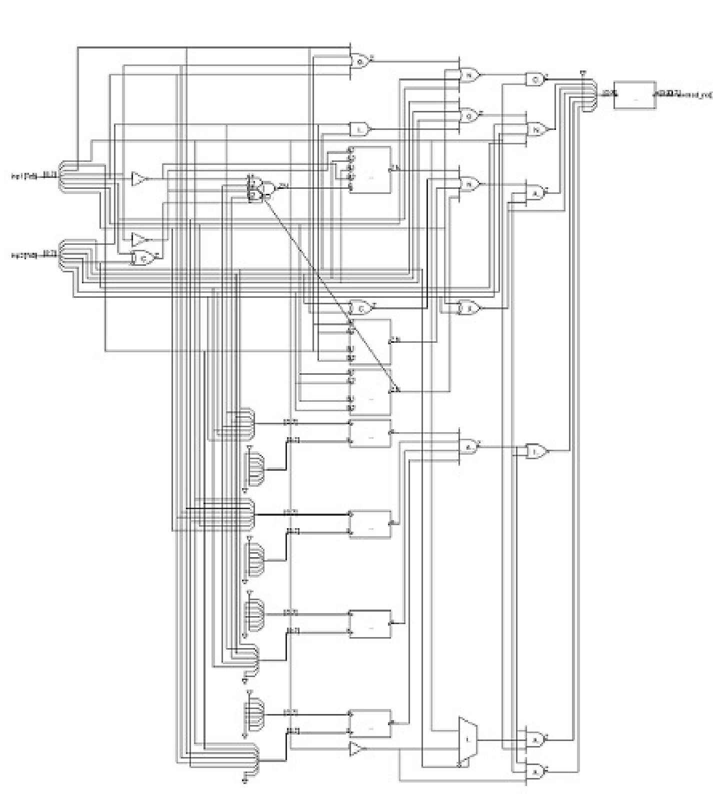

Figure 6.22: $8 - bit$ Nikh SS

6.7.2 RTL Results

Table 6.7: Logic

# bits	Area(nm^2)	Power(nW)	Delay(ps)
8	51	3605.177	190.60
16	88	5701.256	209.50
32	165	9414.358	367.30
64	307	17494.020	382.40
128	608	33077.098	775.50

6.8 CLA Blocks

6.8.1 RTL Analysis

Table 6.8: CLA Analysis

CLA	Area(nm^2)	Power(nW)	Delay(ps)
cla2	10	1222.249	76
cla4	27	2740.212	112
cla8	82	7792.435	189
cla16	115	12547.3	252
cla32	232	26374.588	359
cla64	464	50265.548	407
cla128	938	104861.319	511
cla256	1882	210065.503	568

6.9 Test Bench Output

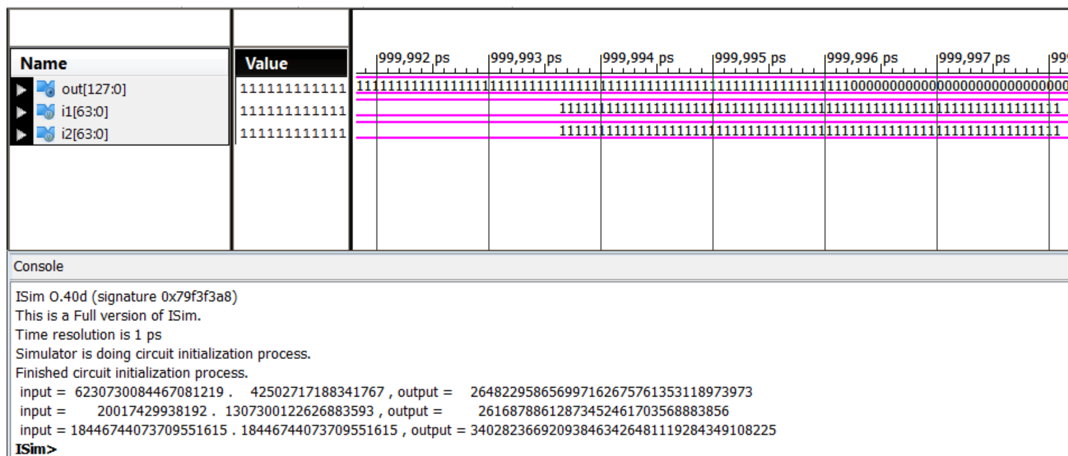

Figure 6.23: $64 - bit$ Vedic UT

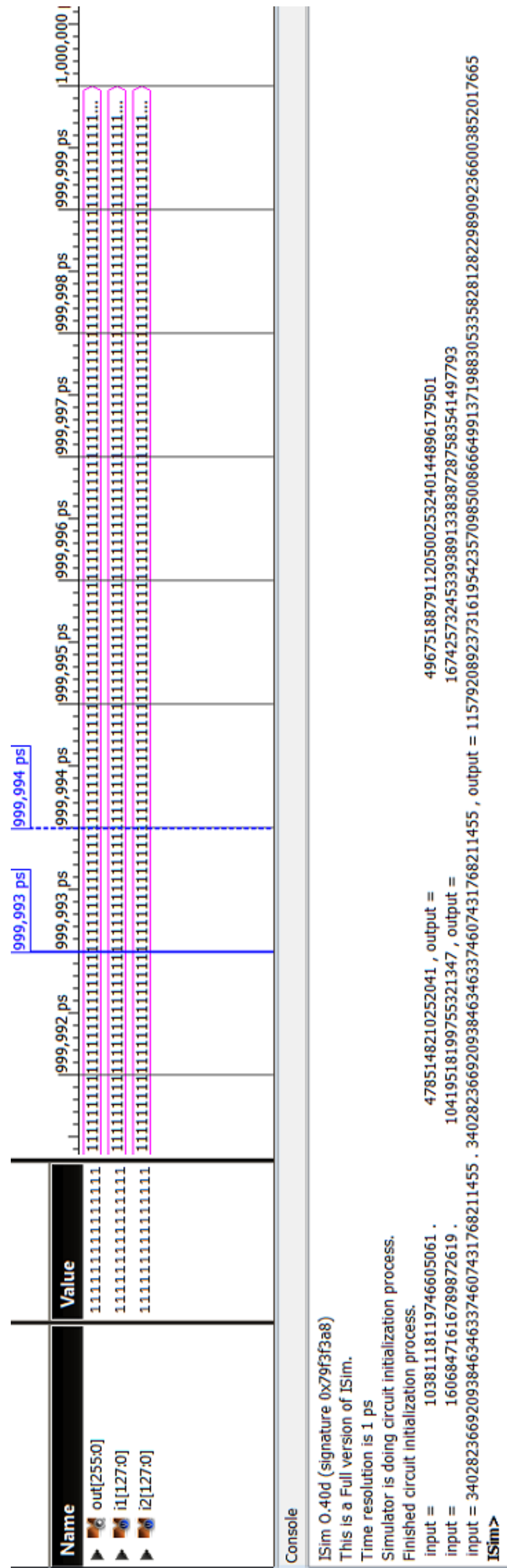

Figure 6.24: $128 - bit$ Vedic UT

Chapter 7

Inferences

The schematics and the RTL results for each of the blocks were given in the previous chapter. However, just the results alone are insufficient to gain a complete understanding of the project. Rather, knowing what blocks/what codes/what logic caused these results and how they caused the results to come, can help us to evaluate it as well as predict the outcomes for optimization if the project is expanded to a higher scale in the future.

Not only this, but a complete analysis of the results can be made only by comparing each block with the other as well as comparing with the existing literature. These comparisons can yield insight into knowing which module would be most optimum in which situation as well as for estimating the efficiency and the drawbacks.

7.1 Comparison – Vedic UT, Vedic Square & Design D

Power Consumption

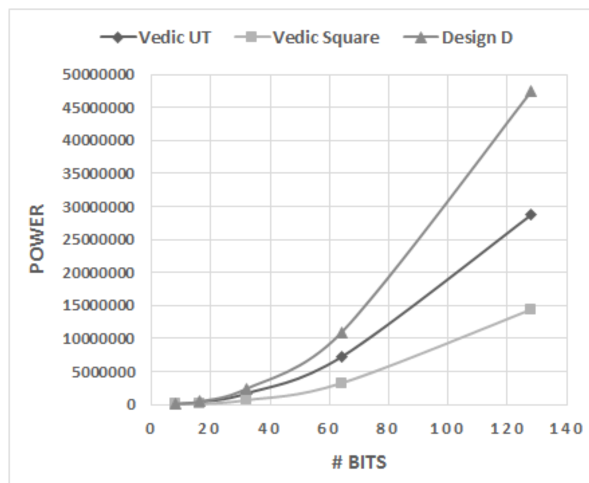

Figure 7.1: Power Consumption in nW

Area occupancy

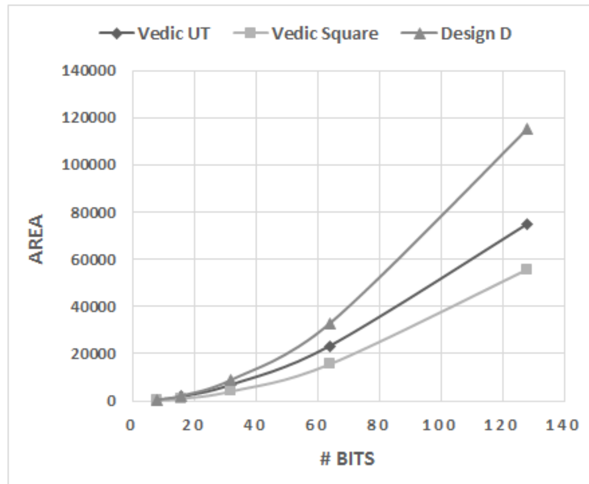

Figure 7.2: Area occupied in nm^2

Delay

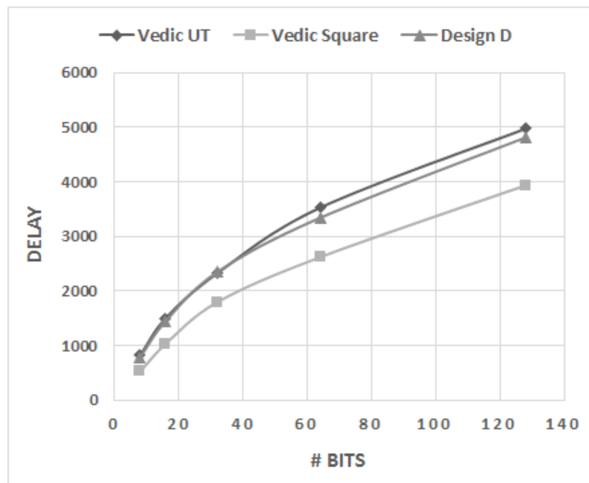

Figure 7.3: Worst Path Delay in ps

It is clear that the Vedic Square is the most optimum – having a clean, almost linear graph with a very low slope. It is not surprising too that values increase less rapidly with the increase in number of bits as compared to the other two, for after all, the input size is just a fraction of what it is for the rest.

It is observed that the increase is more for Design D in both Area and Power as compared to Vedic UT when the number of bits increases. However, for the Delay, it is seen that Design D is slightly better than Vedic UT. Thus it is more advisable to go for Design D only when the speed is of primary importance and power dissipation and area occupancy are of no concern. Then, better delay will be obtained, however slight it may be.

7.2 A look at the 3 cases of Nikhilam

With the power given in nW, Area in nm^2 and the delay in ps.

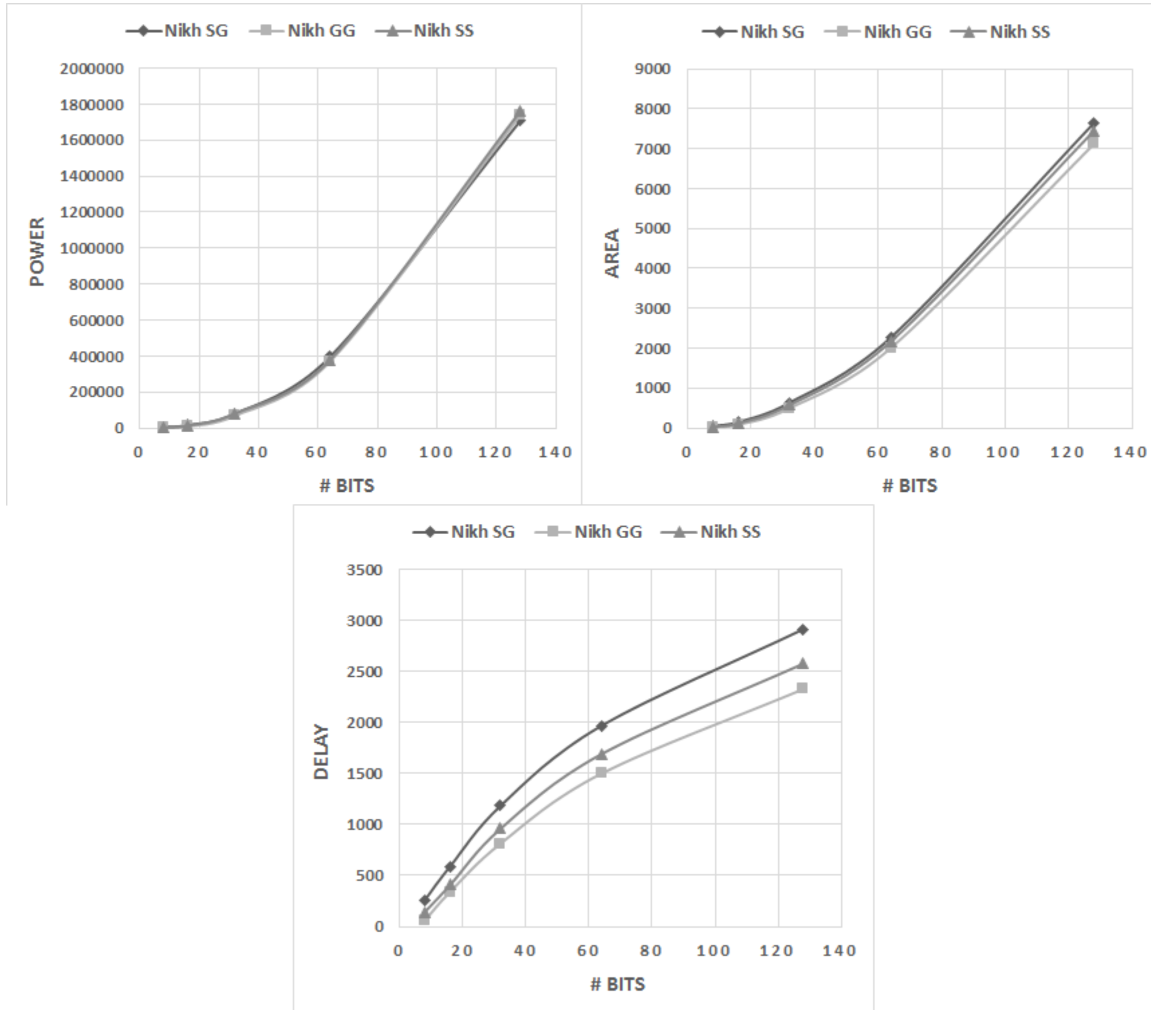

Figure 7.4: Comparison between the 3 cases of Nikhilam

All the 3 cases of Nikhilam have almost the same Power Consumption curve. The Area curves are slightly different while maximum variance can be observed in the Delay curves. When both the inputs are larger than the base with the Nikhilam threshold (Nikhilam GG), the delay is the least, followed by Nikhilam SS which requires both the inputs to be below the base within the said threshold. Nikhilam SG which deals with one input being larger and the other being smaller than the base has the maximum delay associated with it. This can be put to use in application specific design of multipliers.

Another important fact the graphs of Nikhilam yield is that **all the 3 cases have the same power**, more or less. This has a huge impact while doing the power analysis, since all the 3 cases can be considered as one and need not be dealt as separate cases. This will simplify the power analysis tremendously.

7.3 Analytical Comparisons with the Vedic UT

Table 7.1: Vedic UT to Vedic Square: % reduction

# bits	Power(%)	Area(%)	Delay(%)
2	60.3	60	37.6
4	58.1	50.7	29.4
8	68.8	58.8	36.4
16	66.8	50.7	36.4
32	60.7	41.4	22.9
64	55.2	33.4	25.7
128	49.9	25.7	20.9

Table 7.2: Vedic UT to Design D: % reduction

# bits	Power(%)	Area(%)	Delay(%)
8	-23.2	-16.1	4.3
16	-23.5	-18.1	3.6
32	-42.0	-29.5	-0.9
64	-51.4	-40.8	5.3
128	-65.0	-53.7	3.2

Table 7.3: Vedic UT to Nikhilam (maximum): % reduction

# bits	Power(%)	Area(%)	Delay(%)
8	92.6	88.5	67.8
16	735.3	91.9	60.6
32	95.2	91.9	60.6
64	94.5	90.3	49.0
128	94.0	89.8	41.3

The above tables indicate the exact percentage of improvement of the designs from the Vedic UT. Since the values of Design D are negative, it can be inferred that

The performance of

$$Nikhilam\ >\ Vedic\,Square\ >\ Vedic\,UT\ >\ Design\ D$$

7.4 Analysis of the Logic Block and CLA Adder

7.4.1 Logic Block

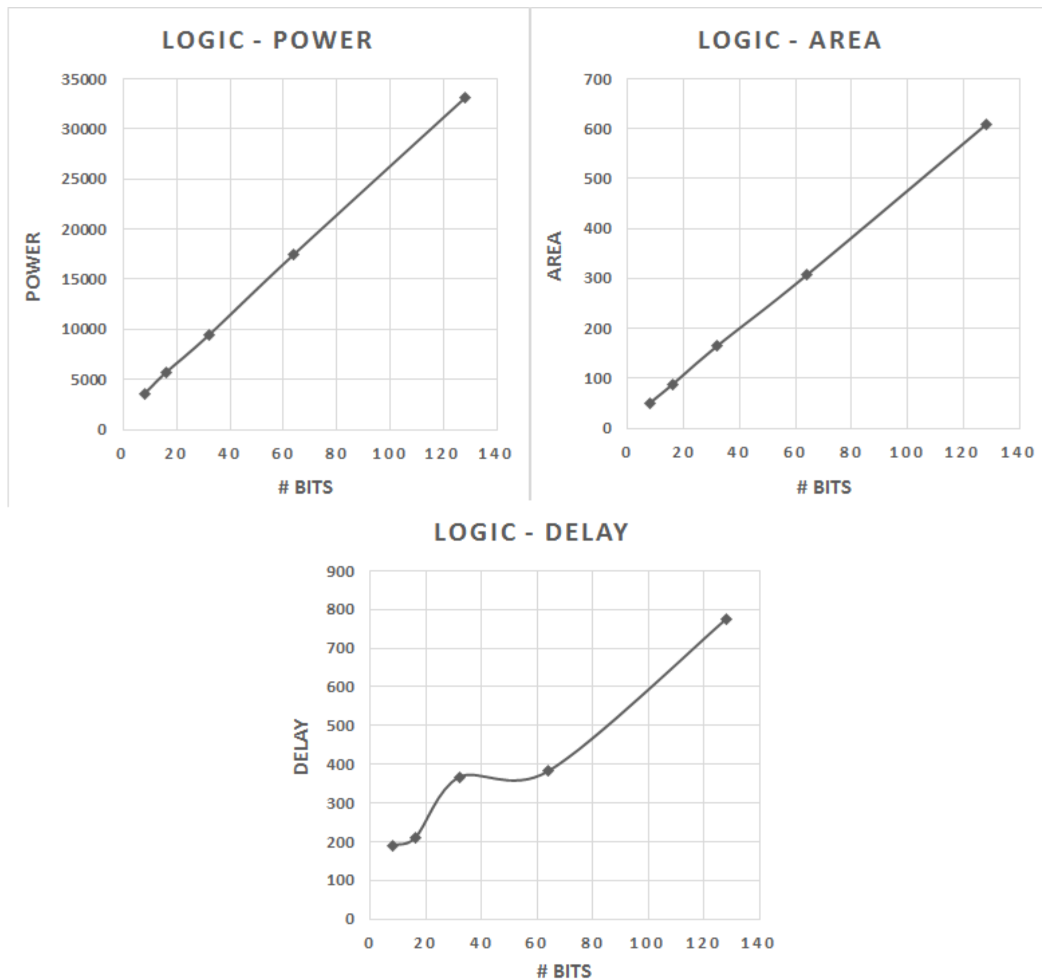

Figure 7.5: Logic Modules – Variation of Parameters

As can be seen, there is a linear relationship between the Power and number of bits as well as between the area and number of bits. The delay is almost linear with just one outlier. This fact can be used for estimating the values for larger logic blocks.

This linearity probably comes from the fact that all the logic blocks have the same dependence on comparators and the comparators are built linearly.

7.4.2 CLA Adder

Another block that shows linearity is the CLA adder.

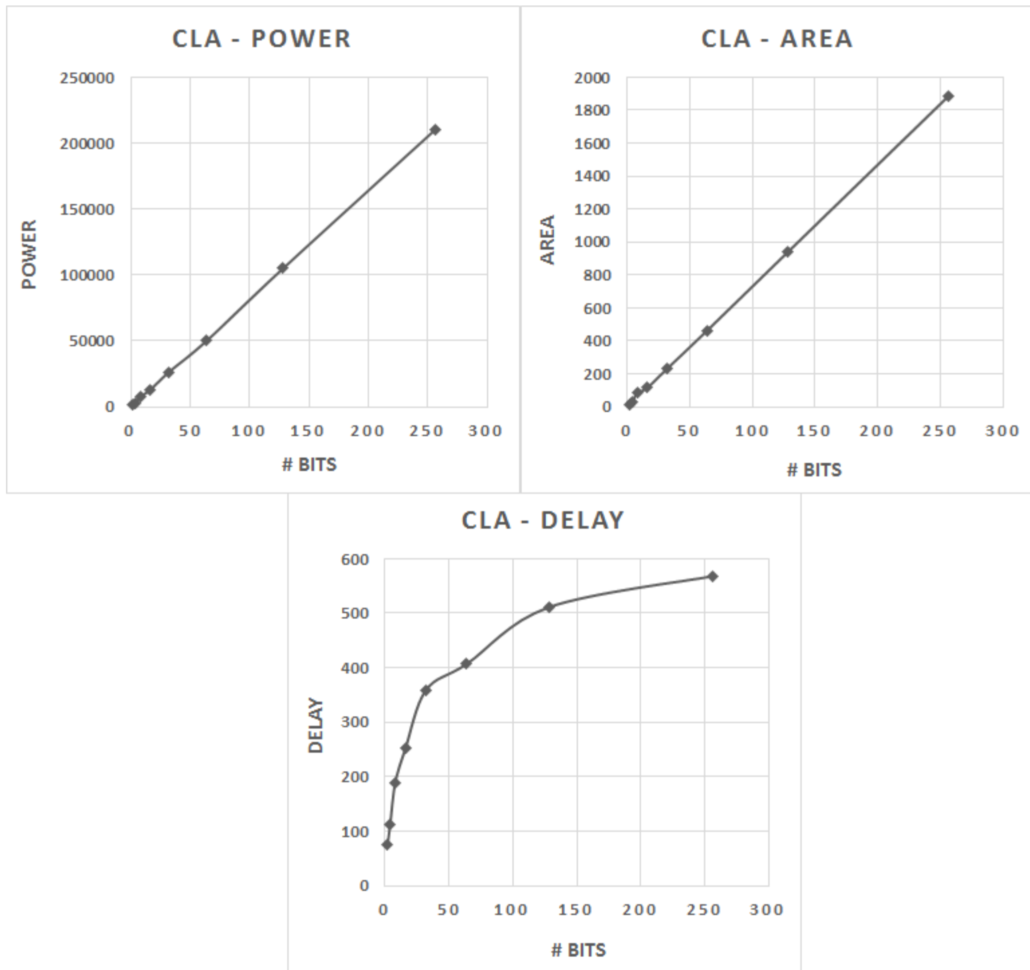

Figure 7.6: CLA Adder – Variation of Parameters

The power and area again follow a linear relationship with the number of bits while the delay is completely non-linear. This should not be surprising as we proved that CLA adder has a logarithmically growing delay. Hence, for lower power applications of the multiplier, some other adder should be chosen but for most of its part, the CLA is an excellent choice.

Chapter 8

Comparison with Existing Work

On comparison with other reported multipliers, it is seen that even the Vedic UT, the maximum valued block in ***Sampoornam*** is more efficient. Hence, it can be concluded that the ***Sampoornam multiplier*** is very efficient.

Table 8.1: As reported in Literature

# bits	Power(uW)	Area(nm^2)	Delay(ps)	Reference
8	2342	13555	2342	Proposed Design [19]
8	2654	22674	2816	Array [19]
8	1688	12374	2632	Booth [19]
8	-	-	17995	Vedic [13]
8	-	-	15416	Vedic [14]
16	-	-	22604	Vedic [14]
32	-	-	35760	Vedic [14]
64	-	-	44870	Vedic [14]
8	-	-	18699	Nikhilam [17]
16	-	-	20094	Nikhilam [17]
32	-	-	24075	Nikhilam [17]
64	-	-	32816	Nikhilam [17]
8	168000	-	26825	Array [4]
8	99000	-	23079	UT [4]
8	148000	-	27878	Nikh [4]
16	250000	-	53276	Array [4]
16	118000	-	41350	UT [4]
16	118000	-	51323	Nikh [4]
32	382000	-	107128	Array [4]
32	315000	-	72332	UT [4]
32	315000	-	90747	Nikh [4]

- $\longrightarrow$ Data not given.

Table 8.2: Vedic UT

# bits	Area(nm^2)	Power(uW)	Delay(ps)
2	10	0.954171	62.8
4	71	7.782601	340
8	454	63.223888	814.90
16	1913	360.307407	1500.20
32	6909	1688.282769	2321.50
64	23346	7212.137495	3522.10
128	75013	28739.302317	4970.80

As can be seen from the tables, the Vedic UT (and hence the ***Sampoornam***) multiplier are more efficient than those reported in the literature.

With the 21 multipliers compared here, only one $64 - bit$ bit multiplier is reported. Its delay, 32.82 ns is even more than the $128 - bit$ Vedic UT's delay of 4.97 ns. That is almost a 84.72% reduction for a higher order multiplier. Similarly, there's a steep change from the existing multipliers to the proposed multiplier.

Thus we can conclude that the multipliers designed in this report are more optimum than the current multipliers.

Chapter 9

Power Analysis

The disadvantage with performing the power analysis in software, like Cadence here, is that they won't give accurate results as will occur in real life in a few cases. One such case is when many submodules are present but only one of them should be ON at a time based on individual ENABLE pins dependent on the input values. Since **conditional instantiation** of a module is not possible in Verilog, the power as calculated by the software will invariably assume all the submodules are ON at the same time and thus give an exorbitantly high value.

However, this is an inaccurate result. The ***Sampoornam*** multiplier precisely falls under the above category. And power analysis is a must for ***Sampoornam*** in order to judge its performance.

A novel method of analysis is proposed based on mathematical principles of probability. Since the multiplier is now input dependent, consider the following mathematical model presented below.

Let the submodules be $A_1, A_2, \ldots, A_m$ with power as given below.

Table 9.1: Submodules & their Power consumption

Module	Power
A_1	P_1
A_2	P_2
.	.
.	.
A_i	P_i
.	.
.	.
A_m	P_m

Let the range of inputs for the module be denoted by $range = [p, q]$ and let

$$|range| = R \tag{9.1}$$

Let the inputs be denoted by x, y.

Let the range (set of inputs for which the submodule should be ON) of each submodule A_i be given by S_i.

And let p_i be the probability that $x, y \in S_i$.

Note,

$$S_1 \cup S_2 \ldots \cup S_m = range \tag{9.2}$$

$$\sum_{i=1}^{m} p_i = 1 \tag{9.3}$$

Now the actual power dissipated contributed by the submodules can be found by

$$\begin{aligned} P_{mod} &= \sum_{i=1}^{m} p_i.P_i && (9.4) \\ &= p_1.P_1 + p_2.P_2 + \ldots + p_m.P_m && (9.5) \end{aligned}$$

Adding the power dissipated by the logic block, we get

$$\boxed{P_{net} = P_{mod} + P_{logic}}$$

9.1 Power analysis – Sampoornam

Considering the 3 cases of Nikhilam as one, since their power consumption is almost the same, we have the modules as Vedic UT, Nikhilam and Vedic Square (say). If Design D, Left Shift, or any other modules are to be added, they can be done by following the same steps as shown below.

Table 9.2: Submodules – Sampoornam

Module	Power
A_1	Vedic Square
A_2	Nikhilam
A_3	Vedic UT

Let the inputs be $n - bit$ in size. Let the threshold size be k bits.

Probabilities that the inputs belong to a module are given by,

p_1 = prob (inputs $\in$ Vedic Square set)

p_2 = prob (inputs $\in$ Nikhilam set)

p_3 = prob (inputs $\in$ Vedic UT set)

For Vedic Square, to find the probability when the inputs are to be equal, consider that you are selecting out out of 2^n first. The probability of doing so is 1. Now the probability of picking the same number chosen before again implies now the choices are just one number, i.e, that which was picked before. So, the probability is given by

$$p_1 = 1.\frac{1}{2^n} = \frac{1}{2^n} \tag{9.6}$$

For Nikhilam, the number of favourable outcomes is given by $2(2^k - 1)$ for one input. The total number of outcomes are 2^n. So the probability is given by,

$$p_2 = \left(\frac{2(2^k - 1)}{2^n}\right)^2 = \left(\frac{2^k - 1}{2^{2n-1}}\right)^2 \tag{9.7}$$

For Vedic UT, finding probability is easy. From eq (8.3), it is given by

$$p_3 = 1 - p_1 - p_2 \tag{9.8}$$

After this, the parameters are substituted in eq (8.4) and P_{net} is then calculated by adding P_{logic}

For a $16-bit$ multiplier, the value comes to **366004.256 nW** as compared to **502273.461 nW** given by Cadence RTL analysis.

Chapter 10

Conclusion

A $128 - bit$ digital multiplier is built stage by stage starting with a $2 - bit$ and a $4 - bit$ multiplier. The Vedic algorithms – Urdhva Tiryakbyam and Nikhilam are effectively utilized in order to increase its efficiency. Along with the Vedic principles, the Karatsuba-Ofman algorithm is implemented to scale a higher order multiplier from a lower order multiplier.

Many designs based on known Vedic algorithms for decimal numbers are adapted for digital systems and implemented with many innovations. To summarize, the major designs are the **Vedic UT** – based purely on the Urdhva Tiryakbyam, **Vedic Square** – which exploits the ease of squaring digital numerbs using Vedic principles, the **Thresholding Nikhilam** which uses the Nikhilam algorithm with a threshold, the **Successive Nikhilam** which recursively uses Nikhilam to reduce the multiplication to just a $1 - bit$ AND operation and **Design D** – which is based on a famous algebraic manipulation.

A unique integrated multiplier – the ***Sampoornam*** multiplier is proposed and a separate logic block is built to make use of all the modules mentioned above. These various designs proposed above are most optimum for a specific set of inputs particularly. This has been incorporated into ***Sampoornam*** in order to make it a complete multiplier. The features of ***Sampoornam*** include the presence of a logic block to determine which design is to be used for a given set of inputs along with well optimized multiplier designs which makes it unique.

Each module of the integrated multiplier is compared with other modules designed as well those reported in literature. A detailed analysis has been done and parameters such as time delay, power dissipation and area occupied are found for each case. These are compared with the existing multipliers and the proposed multiplier is found to be more optimum than many of the existing multipliers.

10.1 Future Scope

The multiplier can be made more optimum by using different adders apart from the one proposed using CLA. This is possible because of its modularity. Each multiplier module can be made more optimum by designing a more efficient adder since almost every block depends on an adder. Better designed comparators can also highly optimize the integrated multiplier since the Logic unit makes use of comparators.

The multiplier can be extended for signed numbers (which just involves an XOR operation of the MSBs) as well as for floating point numbers. Globally synchronous, locally asynchronous or globally asynchronous, locally synchronous MAC units can be developed using this multiplier.

This multiplier can be fabricated after making the layout. Also, the results obtained from the RTL analysis can be made better by using more updated technology like 90 nm processes. Like in ASIC design, the multiplier can be customized for each constraint criterion with respect to specific applications.

There is always a huge demand for an optimized multiplier in the industry and it is truly amazing to see that the ancient centuries-old Vedic principles can be used to satisfy the present day demands. More research should be done to uncover these hidden facts which can reshape our future completely.

Bibliography

[1] Ajinkya Kale, Shaunak Vaidya, Ashish Joglekar, A Generalized Recursive Algorithm for Binary Multiplication based on Vedic Mathematics

[2] Devika Jaina, Kabiraj Sethi, Rutuparna Panda, Vedic Mathematics based Multiply Accumulate Unit, 2011 International Conference on Computational Intelligence and Communication Systems

[3] G.Ganesh Kumar, V.Charishma, Design of High Speed Vedic Multiplier using Vedic Mathematics Techniques, International Journal of Scientific and Research Publications, Volume 2, Issue 3, March 2012

[4] Harish Kumar, Implementation and Analysis of power, area and delay of Array, Urdhva and Nikhilam Vedic Multipliers, International Journal of Scientific and Research Publications, Volume 3, Issue 1, January 2013

[5] Harpreet Singh Dhillon and Abhijit Mitra, A Reduced-Bit Multiplication Algorithm for Digital Arithmetic, World Academy of Science, Engineering and Technology 19, 2008

[6] Himanshu Thapliyal, Saurabh Kotiyal* and M.B Srinivas, Design and Analysis of A Novel Parallel Square and Cube Architecture Based On Ancient Indian Vedic Mathematics

[7] Honey Durga Tiwari, Ganzorig Gankhuyag, Chan Mo Kim, Yong Beom Cho, Multiplier design based on ancient Indian Vedic Mathematics, 2008 IEEE International SOC Design Conference

[8] Jagadguru Swami Sri Bharath, Krishna Tirathji, Vedic Mathematics or Sixteen Simple Sutras from The Vedas, Motilal Banarsidas, Varanasi (India), 1992.

[9] Jan M. Rabaey, Low power design essentials

[10] Kabiraj Sethi, Rutuparna Panda, An Improved Squaring Circuit for Binary Numbers, International Journal of Advanced Computer Science and Applications, Vol. 3, No.2, 2012

[11] Manoranjan Pradhan, Rutuparna Panda, Sushanta Kumar Sahu, Speed Comparison of 16×16 Vedic Multipliers, International Journal of Computer Applications (0975 8887), Volume 21 No.6, May 2011

[12] Michael Andrew Lai, 2002, Arithmetic units for a high performance digital signal processor, B.S. (University of California, Davis), Thesis report

[13] Mohammed Hasmat Ali, Anil Kumar Sahani, June 2013 Study, Implementation and Comparison of Different Multipliers based on Array, KCM and Vedic Mathematics Using EDA Tools, International Journal of Scientific and Research Publications, Volume 3, Issue 6

[14] M. Ramalatha, K. Deena Dayalan, P. Dharani, S. Deborah Priya, High Speed Energy Efficient ALU Design using Vedic Multiplication Techniques, ACTEA 2009, July 15-17, 2009 Zouk Mosbeh, Lebanon

[15] Pavan Kumar U.C.S, Saiprasad Goud A, A.Radhika, FPGA Implementation of High Speed 8-bit Vedic Multiplier Using Barrel Shifter, International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 3, March 2013

[16] Prabir Saha, Arindam Banerjee , Partha Bhattacharyya , Anup Dandapat, 2011. High Speed ASIC Design of Complex Multiplier Using Vedic Mathematics, IEEE Students' Technology Symposium, IIT Kharagpur

[17] Ramachandran.S, Kirti.S.Pande, Design, Implementation and Performance Analysis of an Integrated Vedic Multiplier Architecture, International Journal Of Computational Engineering Research, May-June 2012, Vol. 2. Issue No.3, 697-703

[18] Shri Prakash Dwivedi, 2013, An Efficient Multiplication Algorithm Using Nikhilam Method, (arxiv:1307.2731v5)

[19] S. Deepak and Binsu J Kailath, 2012, Optimized MAC unit design, (2012) IEEE EDSSC 2012, IEEE International Conference on Electron Devices and Solid-State Circuits held from 3-5 Dec. 2012

[20] Vinay Kumar, 2009 Analysis, Verification and FPGA Implementation Of Vedic Multiplier With Bist Capability, Thapar University

APPENDIX A

Vedic Sutras

The 16 Vedic Sutras as mentioned in (.....) are as follows

1. ***(Anurupye) Shunyamanyat*** - If one is in ratio, the other is zero.
2. ***Chalana Kalanabyham*** - Differences and Similarities.
3. ***Ekadhikina Purvena*** - By one more than the previous One.
4. ***Ekanyunena Purvena*** - By one less than the previous one.
5. ***Gunakasamuchyah*** - The factors of the sum is equal to the sum of the factors.
6. ***Gunitasamuchyah*** - The product of the sum is equal to the sum of the product.
7. ***Nikhilam Navatashcaramam Dashatah*** - All from 9 and last from 10.
8. ***Paraavartya Yojayet*** - Transpose and adjust.
9. ***Puranapuranabyham*** - By the completion or noncompletion.
10. ***Sankalana vyavakalanabhyam*** - By addition and by subtraction.
11. ***Shesanyankena Charamena*** - The remainders by the last digit.
12. ***Shunyam Saamyasamuccaye*** - When the sum is the same that sum is zero.
13. ***Sopaantyadvayamantyam*** - The ultimate and twice the penultimate.
14. ***Urdhva tiryakbhyam*** - Vertically and crosswise.
15. ***Vyashtisamanstih*** - Part and Whole.
16. ***Yaavadunam*** - Whatever the extent of its deficiency.

APPENDIX B

Cadence Encounter

In this project, the various modules are designed in Verilog using **Xilinx ISE Design Suite 14.3** (Project Navigator and ISim Simulator in particular) and then are synthesized using the **Cadence Encounter** EDA tool called RTL Complier (Cadence 6.1 RC Compiler is the commercial name which uses 180 nm technology). RTL Compiler also gives other necessary information like Propagation Delay (PD), Area occupied by the circuit and Power consumed by the circuit.

Cadence Design Systems, Inc. is an electronic design automation (EDA) software and engineering services company, founded in 1988 by the merger of SDA Systems and ECAD, Inc. For years it had been the largest company in the EDA industry producing software for designing chips and printed circuit boards.

Encounter RTL Compiler is a key technology of the Cadence Encounter digital design and implementation platform. Encounter RTL Compiler offers a unique set of patented global-focus algorithms that perform true top-down global RTL design synthesis to accelerate silicon realization. With concurrent multi objective optimization (timing, area, and power intent) and support for advanced low-power design techniques, Encounter RTL Compiler reduces chip power consumption while meeting frequency goals.

According to Cadence, RTL Compiler, "*with its combination of breakthrough algorithms, efficient data structures, and modern programming techniques*" delivers "*the best speed, area, and power after physical implementation for the most challenging designs. New advanced global synthesis technology further improves these results while delivering even faster run times. At the core of Encounter RTL Compiler is a breakthrough synthesis algorithmglobal focus mapping (GFM). This technique devotes more time to examining the overall solution space to deliver an optimized netlist for meeting the design intent goals throughout physical design*".